TECHNISCHE
UNIVERSITÄT
DARMSTADT

RELIABLE NETWORK SERVICES IN FUTURE INTERNET SERVICE PROVIDER NETWORKS:

Reliable and Efficient Control Plane Applications for Virtualized Data Planes in Software-Defined Networking

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

JEREMIAS GEORG JOHANNES LUCIAN BLENDIN

Erstgutachter: Prof. Dr.-Ing. Ralf Steinmetz
Zweitgutachter: Prof. Dr. Holger Karl

Darmstadt 2018

Blendin, Jeremias Georg Johannes Lucian: *Reliable Network Services in Future Internet Service Provider Networks: Reliable and Efficient Control Plane Applications for Virtualized Data Planes in Software-Defined Networking,*
Darmstadt, Technische Universität Darmstadt,
Jahr der Veröffentlichung auf TUPrints: 2019
Tag der mündlichen Prüfung: 26. November 2018

ABSTRACT

Driven by highly efficient over-the-top content providers, traffic on the Internet is increasing and puts pressure on Internet service providers (ISPs) to increase their efficiency as well. A promising approach to increase the efficiency of ISP networks is software-defined networking (SDN). SDN achieves this by separating the control from the data plane through a network protocol and thereby enabling increased automation and resource efficiency. However, today's SDN-based control planes, consisting of control plane applications and an SDN controller to coordinate the data plane access, do not meet the reliability requirements for services in ISPs networks.

With SDN, network services consist of multiple control plane applications combined in a single control plane. The control path in data plane elements is responsible for processing SDN protocol messages to configure the packet processing pipeline, termed data path. Today's SDN controller designs do not virtualize the control path adequately, i.e., the effects of messages from different applications are not sufficiently isolated. Thereby, misbehaving low priority applications can block the control paths for essential high priority applications in ISP networks such as the network fabric. This lack of isolation can lead to control plane applications to fail unexpectedly and prevent the whole control plane from operating reliably.

To this end, we introduce a novel, systematic resource-oriented approach to characterize the control paths in SDN data planes as well as a virtualization design for throughput aspects of control paths to increase the reliability among control plane applications. Based on these findings, we analyze the requirements of applications to operate on virtualized data planes. Local bottlenecks that only affect a single data plane element can be mitigated by shifting load to a different element. We apply this approach to our network function chaining design and investigate its effectiveness and provide insights on how the application should decide on the specifics of the mitigation process. Global control path bottlenecks affect a complete area of an ISP network. We analyze the interaction pattern that our novel Adaptive Software-Defined Multicast (ASDM) and Adaptive Bit-Index Software-Defined Multicast (ABSDM) designs require to identify such a bottleneck. Furthermore, we show how a global packet matching memory bottleneck can be mitigated by shifting the applications' resource usage from matching memory to data rate. We demonstrate the effectiveness of the ASDM application for mitigating control path resource bottlenecks and thereby making it reliable.

In this thesis, we close gaps in the virtualization of control paths that affect both SDN controllers and control plane applications. Thereby, we enable reliable SDN controllers and propose designs for reliable control plane applications to deliver SDN-based network services in ISP networks.

KURZFASSUNG

Getrieben von großen Inhaltsanbietern wie Google steigt das Datenvolumen im Internet immer weiter an. Dadurch werden Internet Service Provider (ISPs) unter Druck gesetzt, die Effizienz ihres Netzmanagements zu erhöhen. Der Einsatz von Software-Defined Networking (SDN) zur Implementierung der von ISPs angebotenen Dienste ist ein Weg dieses Ziel zu erreichen. ISP Netze stellen jedoch hohe Anforderungen an die Zuverlässigkeit der erbrachten Dienste und damit der SDN Control Plane. SDN Control Planes bestehen aus Steuerungsanwendungen und einem deren Zugriff auf die Data Plane koordinierenden SDN Controller. Diesen hohen Anforderungen werden heutige Control Planes nicht gerecht. Der Hauptgrund dafür besteht darin, dass der Control Pfad, also der Teil von Data Plane Elementen, der für die Ausführung von SDN Protokollnachrichten zuständig ist, nicht ausreichend virtualisiert ist. Dies kann dazu führen, dass der Zugriff auf die Data Plane von für den Netzbetrieb essentiellen Anwendungen durch unwichtige Anwendungen blockiert wird. Dieser Mangel an Zuverlässigkeit ist Grund, warum SDN heute nicht für den Einsatz in ISP Netzen geeignet ist.

Um dieses Problem zu lösen, stellen wir in dieser Arbeit einen ressourcenorientierten Ansatz zur Analyse der Leistungseigenschaften der Control Pfade von SDN Data Planes vor. Der Ansatz wird beispielhaft auf ein State-of-the-Art Data Plane Element angewendet und damit erstmals gezeigt, wie Performanceeigenschaften analysiert werden können. Als ein Ergebnis stellen wir erstmalig ein Verfahren zur Virtualisierung des Einfügens neuer Regeln in SDN Data Planes vor. Auf Basis dieser Erkenntnisse untersuchen wir anhand von zwei repräsentativen Ansätzen, Network Function Chaining und Multicast, wie SDN Control Plane Anwendungen auf lokale und globale Engpässe in Control Pfaden reagieren können. Lokale Engpässe treten dabei auf einzelnen Data Plane Elementen auf. Die präsentierte Network Function Chaining Anwendung reagiert auf solche Engpässe mit einer Verlagerung der Control Pfad Last auf ein anderes Data Plane Element. Eine globale Speicherknappheit in der Data Plane kann durch das vorgestellte Adaptive Software-Defined Multicast System über einen gezielten und steuerbaren Trade-Off zwischen der Nutzung von Speicher in der Data Plane und für die Übertragung benötigte Datenrate umgangen werden. Weiter stellen wir mit Adaptive Bit-Indexed Software-Defined Multicast einen adaptiven Multicast-Ansatz auf Basis des effizienten Bit-indexed Replication Verfahrens vor. Für beide Anwendungen wird untersucht, auf Basis welcher Informationen sie über die Reaktion auf Engpässe entscheiden sollten.

Zusammenfassend zeigen wir in dieser Arbeit, wie die Leistungseigenschaften der SDN Data Plane systematisch untersucht werden und wie die dabei gewonnenen Informationen genutzt werden können, um die Zuverlässigkeit und Effizienz der Diensterbringung zu verbessern, um damit den Anforderungen von ISPs gerecht zu werden.

DANKSAGUNG

Der erfolgreiche Abschluss meines Promotionsvorhabens wäre ohne Hilfe, Unterstützung und Rat von meiner Familie, von Freunden und Kollegen nicht möglich gewesen. Zuerst vielen, vielen Dank an meine Familie und Freunde für ihre Unterstützung nicht nur in den letzten fünf Jahren.

Bei Prof. Dr.-Ing. Ralf Steinmetz möchte ich mich für die Möglichkeit zum Abschluss meiner Arbeit bedanken und für die tolle Arbeitsumgebung in der ich nicht nur gearbeitet und promoviert, sondern auch studiert habe. Bei Prof. Dr. David Hausheer bedanke ich mich für die Betreuung in den ersten drei Jahren meiner Promotion und die Einführung in die Netzwerkcommunity. Dr. Boris Koldehofe, mein direkter Vorgesetzter in den letzten zwei Jahren meiner Promotion, hat mich immer sehr unterstützt und mir das Feedback sowie den Freiraum gegeben hat, um sie erfolgreich ab zu schließen, danke dafür. Vielen Dank auch an Prof. Dr. Holger Karl, dafür dass er sich bereit erklärt und die Zeit genommen hat, Zweitgutachter meiner Dissertation zu sein.

Dr.-Ing. Julius Rückert hat mich durch seine phantastische Betreuung während meiner Abschlussarbeit dazu gebracht zu promovieren und mich auch als Kollege immer tatkräftig unterstützt hat, vielen Dank! Vielen Dank auch an Dr.-Ing. Björn Richerzhagen und Dr.-Ing. Binh Nguyen, die mir als Kollegen und Freunde durch Höhen und Tiefen meiner Promotion geholfen haben. Dr.-Ing. Matthias Wichtlhuber und Dr.-Ing. Fabian Kaup haben mir in den ersten Jahren der Promotion auf gleiche Weise das Leben erleichtert, danke dafür. Den größten Teil meiner Zeit habe ich in den letzten fünf Jahren mit der Arbeit an Projekte für und mit der Deutsche Telekom FMED verbracht. An diesen Projekten habe ich mit Dr.-Ing. Leonard Nobach gearbeitet, mit und von dem ich vieles lernen konnte, vielen Dank dafür. Vielen Dank an die Kollegen von der Deutschen Telekom für die spannenden Themen und die gute Zusammenarbeit: Dr. Hans-Jörg Kolbe, über den ich zur Programmiersprache P4 gefunden habe, Robert Soukup, Georg Schyguda, Yuriy Babenko, Dennis Kusidlo, Dr. Felix Wissel, Dr.-Ing. Matthias Gunkel, Nicolai Leymann, Jochen Appel und an alle anderen.

Das letzte Jahr der Promotion hat dank Ralf Kundel besonders viel Spaß gemacht. Er hat sich nicht nur meine altklugen Sprüche länger angehört als jeder Kollege vor ihm, die Zusammenarbeit mit ihm war zudem noch sehr angenehm und produktiv, danke und großen Respekt dafür. Weiter möchte ich mich bei allen anderen Kollegen bedanken, ohne Anspruch auf Vollständigkeit, bei Dr. Amr Rizk, Rhaban Hark, Tobias Meuser, Nils Richerzhagen, Lena Despres, Manisha Luthra, und allen an dieser Stelle ungenannten. Natürlich vielen Dank an jeden, die den Lehrstuhl am Laufen halten, den ATMs: Frau Scholz-Schmidt, Karola Schork-Jacoby, Frank Jöst, Frau Ehlhardt, Monika Jayme, Thomas Lenz, Britta Frischmuth-Zenker, Jan Hansen und Sabine Kräh. Schließlich noch vielen

Dank meinen Studenten, die mit mir an meinen Themen gearbeitet haben und von und mit denen ich viel gelernt habe, in alphabetischer Reihenfolge: Javier Acosta Corredor, Dennis Albrecht, Sascha Bleidner, Fabrice Bendfeldt, Felix Breidenstein, Tingting Chen, Marvin Härdtlein, Daniel Herrmann, Felipe Villa Arenas, Tobias Volk, Xin Zhang und Patrick Welzel.

Ohne euch alle wären die letzten fünf Jahre nicht so interessant, lehrreich, angenehm und erfolgreich gewesen, es hat Spaß gemacht, danke!

CONTENTS

1 INTRODUCTION

Internet service providers (ISPs) are one of the major building blocks of the Internet. They provide global network connectivity and other essential services to their private and business customers through mobile as well a residential access networks. Driven by technological advances and popular over-the-top (OTT) content, the data rates of their customers are continuously increasing [CVNI17]. This puts ISPs into a unique and challenging position: they offer services at large scales and under intense competition. OTT content providers like Google and Amazon also provide services at large scales but have a better profit and revenue position in the market. This position was enabled by the increasing compute management efficiency since the early 2000s, driven by the widespread adoption of compute virtualization and automation. In contrast to that, management and control cost-efficiency of networking and, consequently, ISPs, has not substantially increased. OTT content providers benefit from the increase in compute efficiency, enabling their massive growth in compute capacity[1][2], while ISPs cannot profit from this development to the same extent.

To alleviate this situation and increase the efficiency in networking, McKeown et al. proposed software-defined networking (SDN) [McK09]. SDN achieves efficiency by logically centralizing and opening up the control plane to innovation through enabling control plane applications to customize the control plane behavior. Recently, ISPs started implementing network services through SDN [Csá+13; ONF17; Nob+17], and in combination with network functions virtualization (NFV) [Pet+16]. NFV is a complementary technology that increases the efficiency in networking by implementing network functions on cost-efficient x86 compute platforms and aims at use cases that are too complex for SDN [NHH16]. Hence, SDN and NFV are expected to significantly increase the management and control efficiency in networking for ISPs.

However, the requirements of ISP networks for SDN control planes are demanding: they require reliability while controlling a complex network as depicted in Figure 1.1. Reliable SDN control planes and their components: controllers and applications, are expected to operate in the face of unexpected events in the data plane and degrade their service gracefully if required. This requirement means that applications must be able to adapt themselves to resource shortages and to continue to operate. In case the built-in reliability mechanisms fail, the control plane must be simple enough to be understood by human network operators so that they can intervene [DPM12].

1 J. Greene. *Tech's High-Stakes Arms Race: Costly Data Centers*. Accessed: 2018-9-21. Wall Street Journal, 2017. url: https://www.wsj.com/articles/techs-high-stakes-arms-race-costly-data-centers-1491557408.

2 H. Liu. *Amazon EC2 grows 62% in 2 years*. Accessed: 2018-9-21. 2014. url: https://huanliu.wordpress.com/2014/02/26/amazon-ec2-grows-62-in-2-years/.

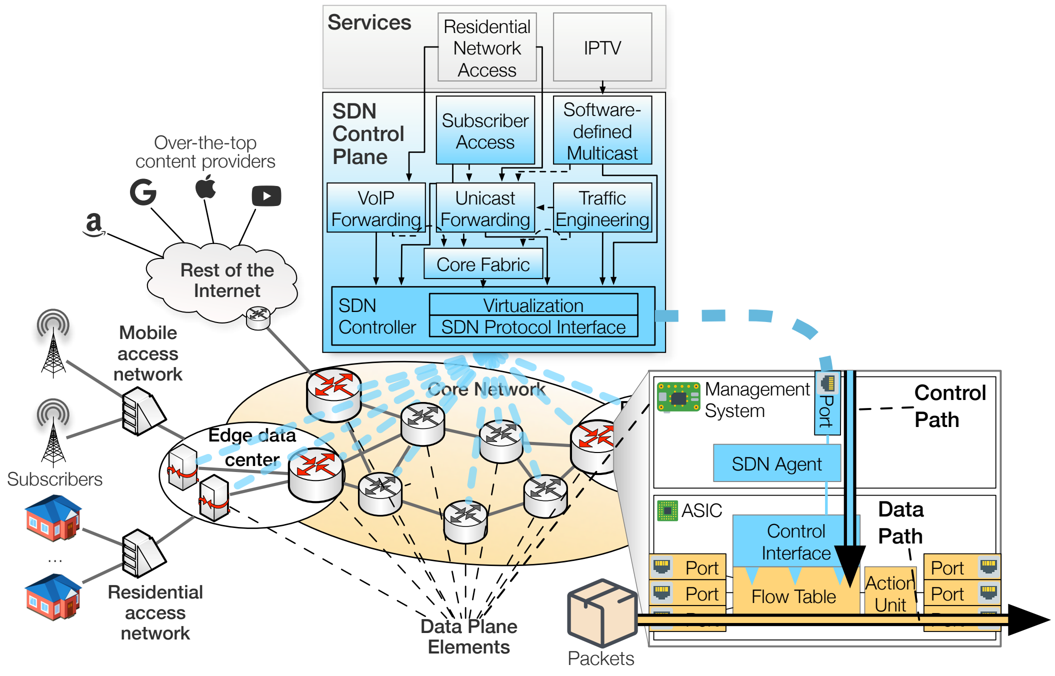


Figure 1.1: An overview of SDN in ISP networks.

However, today's SDN control planes do not provide the required reliability. As depicted in Figure 1.1, in data plane elements, the control path processes SDN control protocol messages to configure the data path that conducts the packet processing. While bottlenecks in the data path are well-understood in both traditional networking and SDN [Zin+14], the understanding of control paths is lacking. Operating multiple control plane applications in a control plane requires the controlled sharing of the control paths of data plane elements as well as isolation between applications, i.e., virtualization to be provided by the SDN controller. However, the control path virtualization in SDN has gained limited attention in academia and industry. Specifically, throughput aspects of the control path performance and their virtualization have been neglected. Some applications are fundamental to ISP networks and their services, such as the *Core Fabric* application that provides connectivity in the core network. Other applications such as multicast are less critical. Hence, the ISP *Core Fabric* application should take precedence over the multicast application in case of performance bottlenecks. If the SDN controller does not provide prioritization, the multicast application can completely block the control path of a data plane element. This, in turn, can lead to the failure of the *Core Fabric* application; resulting in unreliable data plane behavior.

We argue that the effects of overloaded control paths can be mitigated if the SDN controller understands the performance characteristics of data plane elements, can control the data plane load, and provides performance information to affected control plane applications. Therefore, our goal is to ensure that SDN controllers and control plane applications have enough information to react to unexpected situations and thereby operate reliably.

1.1 PROBLEM STATEMENT AND RESEARCH GAPS

Applying SDN to their networks is crucial for ISPs to stay competitive. Reliable SDN controller designs have been proposed, e.g., by Shin et al. and Sasaki et al. [Shi+14; SPA16]. However, the existing designs can neither provide complete control path performance isolation between control plane applications nor do applications receive sufficient information to react to performance bottlenecks reliably. Consequently, today's SDN control planes are lacking the required reliability to operate network services in ISP networks.

We identified two key research gaps that need to be addressed to solve this problem:

Research Gap 1: *A missing understanding of the control path and its virtualization in SDN data planes*

Regarding the first gap, we identified two reasons for the insufficient performance isolation between control plane applications: (1) the lack of information on the control path performance and (2) the lack of performance-related abstractions in SDN protocols [Laz+14].

The first gap is caused by the lack of a systematic approach to analyzing the performance of the control path of SDN data plane elements. The existing literature relies on ad-hoc methods to identify and characterize the performance of SDN data planes and neglects the control path, especially its throughput aspects. This lack of methodology results in an incomplete understanding of the data plane and unrecognized performance bottlenecks in the control path. When SDN controllers fail to identify or virtualize performance bottlenecks in the control path of data plane elements, uncontrolled interference between control plane applications can occur. No approach exists that ensures that SDN controllers can identify and virtualize all relevant parts of the control path.

The lack of performance abstractions in existing SDN protocols is caused by their focus on the functional aspects of the data plane. Therefore, even if control plane designers are aware of performance limitations or bottlenecks in the data plane, existing protocols provide insufficient means to enable the control plane to detect them or react reliably.

Research Gap 2: *A missing understanding of how to design reliable SDN control plane applications operating on virtualized data planes*

The understanding of the impact of data plane virtualization on control plane applications in literature is limited. The performance characteristics of single SDN control plane applications have been studied, e.g. by Rückert et al. and Agarwal et al. [RBH15; Aga+14]. Research on operating multiple control plane applications today focuses on the logical combination of packet processing rules and the isolation of the packet processing in the data path [Sou+14; Jin+15]. While these results are promising, the understanding of the effects of control path bottlenecks on the operation of multiple applications is lacking.

Multiple control plane applications accessing the same virtualized data plane can lead to resource contention. The effect of such contention on control plane applications has not been investigated yet. Applications faced with resource shortages or performance bottlenecks, both permanent or transient, must be able to adapt themselves accordingly to operate reliably. An understanding of the information required by applications and strategies for them on how to react appropriately is needed, but not addressed in literature today.

1.2 RESEARCH GOALS AND CONTRIBUTIONS

The overarching goal of this thesis is to enable reliable network services implemented on SDN control planes. This goal requires that multiple control plane applications operate concurrently and reliably on the same SDN controller. Ensuring control plane reliability requires the SDN controller to isolate the control path between control plane applications completely. Furthermore, applications need the ability to react to unexpected performance bottlenecks or shortages, termed performance events, in the control paths of the data plane. Performance events that affect a single data plane element only, termed local resource events, require different coping mechanisms than shortages that affect the whole data plane of the network domain, termed global resource events.

Based on the above problem statement and the research gaps, we formulate two main goals for this thesis:

Research Goal 1: *Design of a systematic approach to virtualizing the control path of SDN data planes that takes all performance-relevant aspects into account.*

The following two questions need to be answered to reach the first research goal:

RQ 1.1: *How to characterize the control path performance in SDN data planes?*

Analyzing the performance characteristics of data plane elements is challenging because they have been mostly investigated ad-hoc as black boxes in the existing literature. The heterogeneity of the architecture and capabilities of data plane elements requires not only a single data plane element model, but a process to create models for arbitrary data plane elements. To this end, we introduce a new resource-oriented view on the SDN data plane [BH14; Ble+16a]. A systematic approach is presented to identify all control path resources in the data plane and to create a model of their performance interdependences. Furthermore, we provide a method to map the messages of an exemplary SDN protocol to this model. Finally, both approaches are combined and applied to a state-of-the-art SDN data plane element, an Edge-Core AS5712-54X 10GbE switch. Thereby enabling, for the first time, the complete modeling of the control path of such data plane elements.

RQ 1.2: *How to virtualize the throughput aspects of control paths in SDN data planes?*

The design of resource virtualizers needs to reflect the characteristics of the shared resource as well as the requirements of the applications consuming the resource. The challenge is to ensure that new virtualization approaches integrate well into existing SDN protocols. Furthermore, the virtualized component is located in the data plane while the virtualization is implemented on the controller. We find that virtualization approaches for throughput aspects of the control path are lacking. One example for these is the slow performance of the memory interface used for updating packet matching tables on data plane elements, which is likely to cause interference between applications. However, it was not identified as a relevant resource for sharing between control plane applications in the literature yet. To that end, an approach to virtualizing dynamic control path resources is presented. The method is applied to a representative resource: for the first time, the memory interface of a packet matching table of a state-of-the-art SDN-enabled hardware switch is virtualized.

To answer the first research goal, we introduce a systematic approach to characterizing the control path performance of data plane elements. Furthermore, we provide the missing virtualization approach for throughput aspects of the control path of data planes. Thereby, we provide SDN controllers with the means to identify and control all performance-relevant aspects of the control paths of data planes.

With SDN controllers being able to provide reliable access to the data plane, control plane applications need to be adapted to operate on reliable controllers.

Research Goal 2: *Enabling network services to operate reliably on virtualized SDN data planes.*

The following two questions need to be answered to reach the first research goal:

RQ 2.1: *How can control plane applications operate reliably in the face of control path performance events that affect a single data plane element?*

Performance events in the control path require control plane applications to adapt their behavior. To that end, we provide a design space analysis for mitigation approaches for applications. Using resources in a different location in the data plane is one especially useful mitigation approach. However, if this approach is applicable at all, it is challenging because the location is one of the most significant aspects of resources in the data plane. The application of this approach is investigated on the example of an overloaded packet matching memory interface in the context of network function chaining, a service that is crucial to interconnect virtual network functions (VNFs). We show that when relevant information on neighboring data plane elements is available, moving the consumption of control path resources to a different location can increase the reliability of the SDN Function Chaining control plane applications [Ble+14; Ble+15a].

RQ 2.2: *How can control plane applications operate reliably in the face of control path performance events that affect the whole data plane of a network domain?*

Permanent global resource shortages require control plane applications to be designed as resource efficient as possible. Transient global resource shortages leave already efficient control plane applications only limited choices: reducing the resource usage, substituting the use of one resource by another one, postponing the resource usage, or a combination of these approaches. All approaches are challenging to implement because many applications are not designed for this use case yet. First the first time, we investigate the impact of disclosing a global packet matching memory resource shortage in the data plane to control plane applications operating in an ISPs control plane. We show that our implementation of the Software-Defined Multicast (SDM) [RBH15] control plane application, Adaptive Software-Defined Multicast (ASDM) [Ble+15b] is an efficient approach regarding packet matching memory consumption. We demonstrate that ASDM can be adapted for a controllable tradeoff between the consumption of matching memory and network traffic and vice versa through a single parameter. Thereby, the ASDM control plane application can react to global resource shortages of matching memory without reducing its service while keeping operational simplicity. Furthermore, we provide a design that applies the adaptive multicast concept to the recently proposed and highly efficient bit-index multicast method [RFC8279] in our Adaptive Bit-Index Software-Defined Multicast (ABSDM) approach.

The exemplary investigation of control path bottleneck mitigation approaches provides, for the first time, an insight into how control plane applications must be designed to operate reliably on virtualized data planes. We provide solutions to ensuring reliability throughout all planes of SDN: from systematically discovering relevant components in the data plane through the virtualization on the controller up to the control plane applications reacting to resource events reliably. Thereby, we close the reliability gaps that prevented the use of the efficient SDN approach in ISP networks.

1.3 THESIS ORGANIZATION

This thesis is organized as follows: the background and related work are discussed in Chapter 2 and Chapter 3 respectively. Requirements and goals, as well as a systematic approach on how to discover performance relevant resources in an SDN data plane and how to virtualize these resources are described in Chapter 4. Two methods on how performance-related information on the data plane can be used to optimize control plane applications are presented in Chapter 5 and evaluated in Chapter 6. Finally, a conclusion is drawn in Chapter 7.

2 BACKGROUND

We provide the context of this thesis in this chapter. An overview of contemporary ISP networks is given in Section 2.1. The architecture of the building blocks of networks, network devices, is introduced in Section 2.2 both for today's appliances as well as for recently introduced software-oriented architectures. Finally, we provide an overview of the SDN approach to network management in Section 2.3.

2.1 INTERNET SERVICE PROVIDERS AND NETWORK SERVICES

In this section, we present the fundamental aspects of ISP networks. A more comprehensive view on this building block of the Internet with regards to technology is provided by Doverspike et al. [DRC10] and with regards to topology and traffic by Betker et al. [Bet+14].

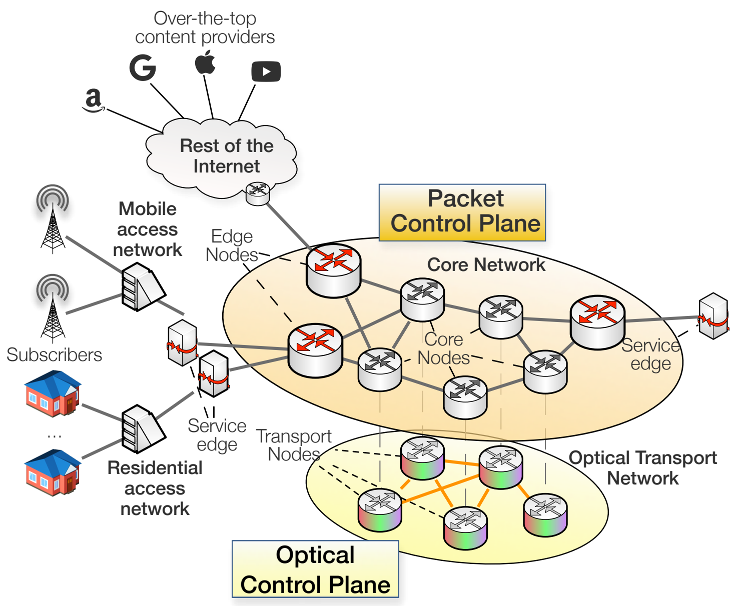


Figure 2.1: Schematic view on an ISP network

ISPs provide Internet access to their residential and mobile customers as depicted in Figure 2.1. Therefore, their networks are designed to bring network access to geographically distributed locations. The main parts of ISP networks are the core network, the network edge, and the access networks. The core network spans vast geographic distances to interconnect geographically distributed edge networks as well as other parts of the global Internet. Edge networks consist of one or more edge data centers from where access networks distribute connectivity to individual subscriber locations. The networking technology used in core and edge networks today are often Ethernet and Internet Protocol (IP). The access network mostly relies on different technologies that enable cost-effective connectivity for individual subscriber locations. Therefore, access networks can be understood as mostly passible packet pipelines. The services are implemented in edge or core locations of ISP networks. The part of the edge network that faces the customers is termed the service edge and represents the boundary of the IP part of the network.

An optical transport network provides the long-range network links for the core network. These networks could be called software-defined today for their use of remotely configurable equipment such as reconfigurable optical add-drop multiplexers (ROADMs). ROADMs enable configurable wavelength switching, thereby enabling optical path switching in optical transport networks. This approach allows ISPs to provide arbitrary network links for the IP-based core network.

Other parts of the Internet, including OTT content providers, are interconnected at edge nodes. The depiction shows one example link to the rest of the Internet. Actual ISP networks have multiple, geographically distributed peering points to exchange traffic.

The primary services offered by ISP networks are Internet access for private and corporate customers and virtual private networks for corporate customers. Services are directly offered to customers and are at least partly commercial entities while network services refer to the technical part of the service delivery. Today, services are implemented by network services using the fully distributed control plane of traditional network equipment. To that end, the core network often operates a separately managed control plane based on, e.g., Multi-Protocol Label Switching (MPLS). The edge locations are managed separately to implement services, e.g., network access through a broadband network gateway (BNG). The core network control plane provides connectivity for all connected locations and thereby is the central distribution platform and a core element of ISP networks. Add-on network services are implemented using network services spanning multiple of those management domains. IP multicast, e.g., used to provide Internet Protocol Television (IPTV), spans the management domains of the core and the edge network.

An inherent hierarchy exists between the network services in ISP networks [Roj+18]. Some applications are fundamental to every service provided, such as the core network while other applications such as multicast are less critical. This means that the core network should take precedence in case of interference or configuration mismatches.

Finally, the importance of the core network and the optical transport network for the ISP business means that their operators are very conservative. New technology is only slowly adopted, and some ISPs require that human operators have the understanding and the ability to override the network management system at all times to ensure its reliability.

2.2 NETWORK DEVICE ARCHITECTURES

Network devices are the key element of networks. We give an overview on how hardware-appliances are built today in Section 2.2.1 as well as the on the recent software-based approach to network data planes called NFV in Section 2.2.2.

2.2.1 *ASIC-Based Appliances*

A network function defines a specific, well defined functional block in a network. Example for network functions are routers, firewalls, deep packet inspection, or network address translation. Traditionally, each of these functions has been implemented in a dedicated network device, an appliance that combines general purpose hardware, accelerator hardware such as application-specific integrated circuits (ASICs), and proprietary software as depicted in Figure 2.2.

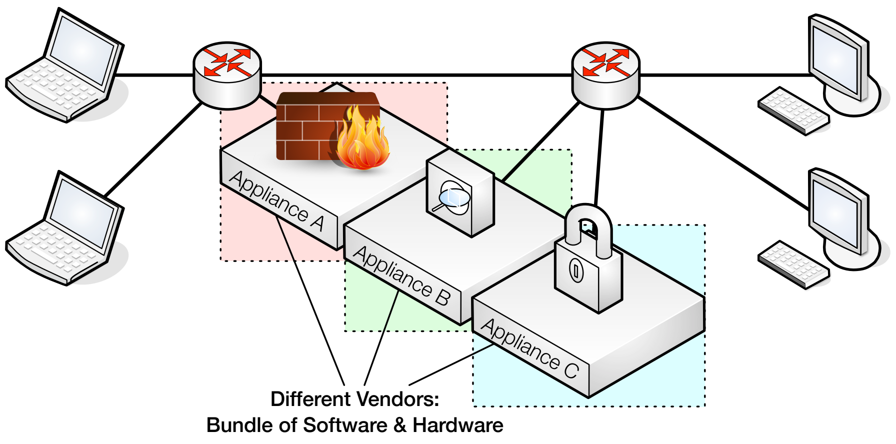


Figure 2.2: Appliance-based networking.

The most widespread devices in networks, routers, and switches, are a representative class of devices that we will use to discuss the architecture of network appliance devices. Figure 2.3 depicts a typical device architecture. The packet processing is mostly conducted in specialized, proprietary hardware devices, often an ASIC but sometimes also network processing units (NPUs) or field-programmable gate array (FPGA). The management system is connected to the packet processing hardware and provides a control interface to the hardware through a standard embedded computer. The idea behind this approach is that the high-throughput part of the device is implemented by specialized, proprietary hardware while the control part, which is less performance

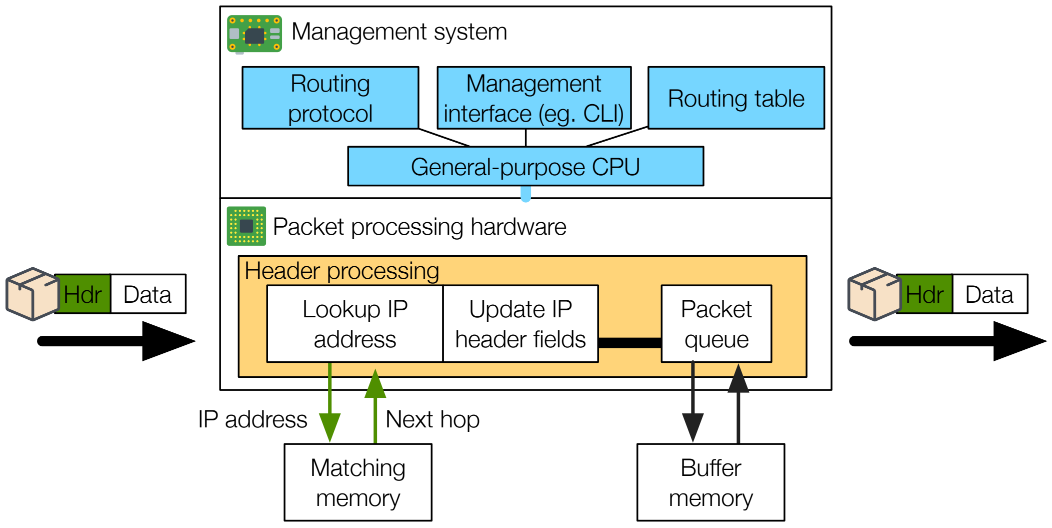


Figure 2.3: A schematic representation of a typical router appliance architecture (adapted from [McK03]).

critical is implemented through a cost-efficient computer and accompanying software. Routing and command line interfaces are typical software that is operated on the management system. Furthermore, a driver to access the hardware device is provided there as well.

Table 2.1: Overview of memory types and their lookup characteristics (adapted from [PV11]).

Technology	Match type	Access time [ns]	Max. size	Cost [$/MB]	Power [W/MB]
TCAM	content, ternary	4	~20Mb	200	15
SRAM	address, binary	0.45	~210Mb	27	0.12
RLDRAM	address, binary	15	~2Gb	0.27	0.027
DRAM	address, binary	55	~10GB	0.016	0.023
SSD	storage only	1,0000	~10TB	0.003	0.00001

As depicted in Figure 2.3 there are two parts of the packet header processing pipeline that rely on memory: the IP address lookup to determine the next hop for a packet and the buffer memory. Both types of memory have different usage patterns but have similar requirements: to achieve a high packet throughput of, e.g., 1 Bpps[1] low access times are required. Table 2.1 lists the main memory technologies available today. All memory types can be used for storing data, which means that a bit pattern is stored at a specific address. This access type is required for buffer memory, which is why often SRAM is used for this task. However, for looking up addresses, the inverse process is required, i.e., looking up a bit pattern and getting an address in return. This type of

1 Edge-coreE. *AS5712-54X 10GbE Data Center Switch Datasheet*. Datasheet EC-DS-0118-07.

memory is called content-addressable memory (CAM) and exists in two variants: binary content-addressable memory (BCAM) and TCAM. The latter is often used to implement IP forwarding lookups for its high speed and ability to conduct ternary matches, which are partial matches on bit patterns. However, its drawback is its small size as well as high costs and power consumption. This is one of the reasons why lookup memory is often scarce in data plane elements.

2.2.2 *Network Functions Virtualization*

The concept of NFV was proposed by the European Telecommunications Standards Institute (ETSI). This introduction relies on the ETSI terminology [ETSI18].

While the traditional device-oriented approach to networking works for well-defined network functions, this approach is inefficient and expensive for less standardized functions. The inefficiency is caused by the fact that each of the devices comes with its own hardware, which inevitably leads to low utilization. Except for the functional interface and the physical interface, there is usually no standardization involved. The high costs are caused by the fact that often each device comes from a different vendor, has a different administrative interface, and requires a separate support contract.

The idea of NFV is to implement network functions on the same hardware platform: standard x86 servers. To that end, an approach and terminology for virtualizing network functions are introduced. VNFs are software implementations of a network functions. Today, the implementations are often provided as virtual machines that operate on a server that provides the required virtualization facilities. The server is termed NFV infrastructure and can host serveral different VNFs instances as depicted in Figure 2.4. Note that the hardware platform for VNF is now standardized. Network connectivity is

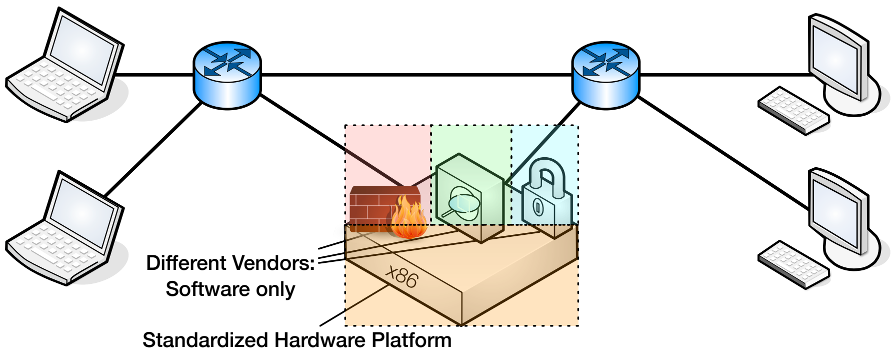


Figure 2.4: Network functions virtualization.

provided through the network interface cards (NICs) of the NFV infrastructure. They are connected to a virtual switch that interconnects all virtual machines through virtual network interfaces. As proposed by the ETSI [ETSI13], the concept includes and focusses on the management of VNFs to increase efficiency and reduce costs.

NFVs infrastructures are commercial off-the-shelf (COTS) servers hosted in a data center. Data centers are organized in racks, for which one or more top-of-rack (ToR) switches provide connectivity to the data center fabric. Both the virtual switches and the data center switches are often SDN-enabled. The NFV management platform relies on SDN to control both types of switches. The need for SDN is caused by the requirement to host different virtual network for different services or customers.

2.3 SOFTWARE-DEFINED NETWORKING

Traditional networking relies on a completely distributed control plane. Each data plane element hosts its own part of the control plane as depicted in Figure 2.5. Standardization only extends to the protocols used to communicate between the data plane elements.

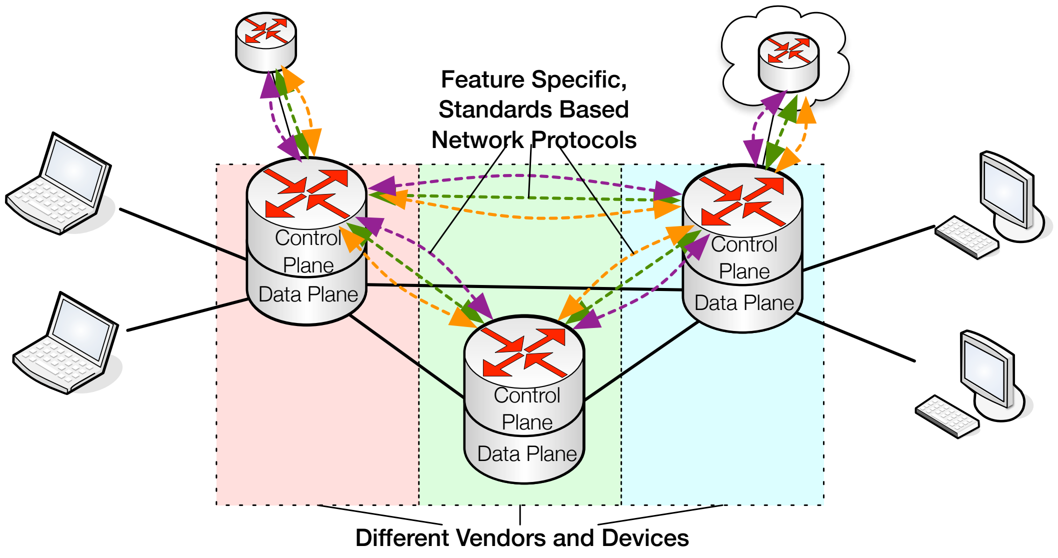


Figure 2.5: Device-oriented networking.

SDN completely separates the control plane from the data plane through an SDN protocol that enables a logically centralized SDN controller to govern the data plane elements. Figure 2.6 depicts the schematics of the SDN architecture. The standardization does not include all parts of the system: the SDN protocol standardizes the communication between the control plane and the devices; still, existing standard protocols are used between different control planes. One example for such a protocol is OpenFlow [ONF15]. OpenFlow is the main SDN protocol used in industry and academia, which is why we base our introduction into SDN and parts of our terminology on it. The control plane is now logically centralized and does not have to be hosted on the individual data plane elements anymore. The term logically centralized refers to the fact that while the control plane is still a distributed system, this fact is hidden from the data plane elements and

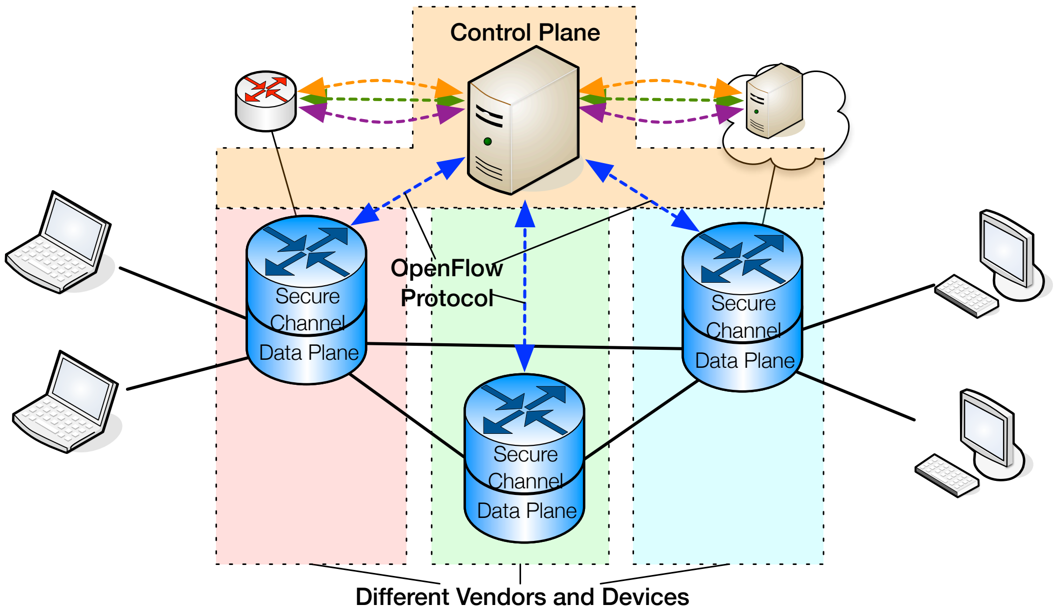


Figure 2.6: Software-defined networking.

the SDN applications operating in the control plane to enable functional abstraction in networking. The part of a network that is governed by an SDN control plane is referred to as SDN network domain.

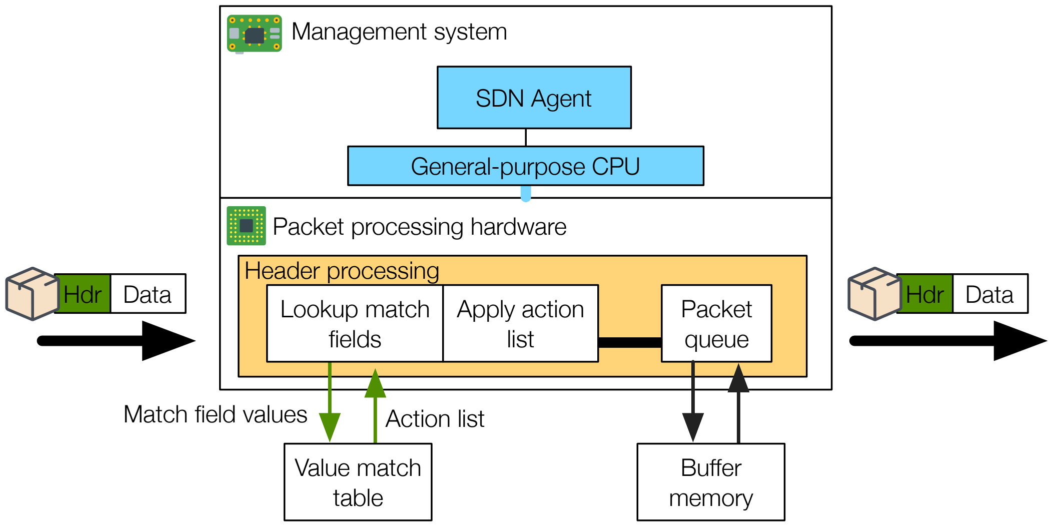


Figure 2.7: A schematic representation of a typical SDN data plane element architecture (adapted from [McK03]).

The management system of the individual SDN data plane elements only host an SDN agent software as depicted in Figure 2.7. Everything else is operated on the centralized control plane. However, the packet processing ASICs have not fundamentally changed, as can be seen in the depiction. The usage of certain hardware features has been generalized, but the fundamental concepts, and thus the hardware, are still the same. Specifically, OpenFlow was designed using the match type of TCAM as the match field lookup specification. Therefore, the central abstraction in OpenFlow to specify the data plane behavior is the match field lookup table, referred to as flow table by OpenFlow. The packet processing is programmed by installing flow entries in the flow table that match for flows of packets and instruct the hardware to apply a given list of processing instructions to them, referred to as action list.

We refer to the packet processing part of the data plane elements as the data path. The path SDN protocol messages take through the management system to affect the data path configuration is referred to as control path.

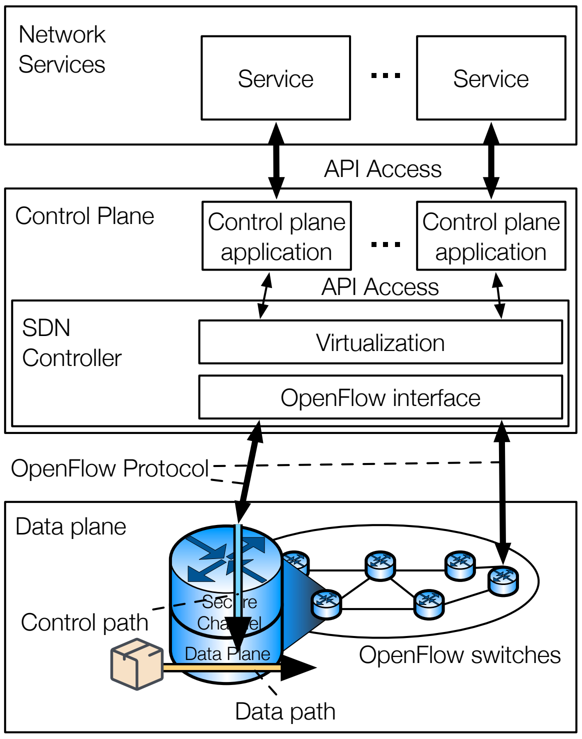


Figure 2.8: A schematic view on the SDN control and management architecture terminology used in this thesis.

A control and management architecture for SDN-based networks has been proposed by the Open Networking Foundation (ONF) the body governing OpenFlow. We already used the terms data plane and control plane without introducing them: the data plane is the network of devices that are responsible for the packet processing. The control plane is the, in the case of SDN logically centralized, part of the system that runs the logic and instructs the data plane elements how to behave.

The logically centralized nature of SDN control planes makes it easier to implement custom control plane behavior. While in traditional networking new behavior requires a new protocol and the corresponding standardization process, with SDN the new behavior only depends on the SDN controller it is implemented on. The ONF distinguishes between a controller plane and an application plane. We follow this concept but refer to the combination of controller and application plane as the control plane. Control plane applications are implemented on top of a controller that provides the Application Programming Interface (API) to access the data plane, topology discovery, virtualization for parallel access of multiple applications and other services. Our view on the SDN control and management architecture is depicted in Figure 2.8.

In SDN, network services are created by combining control plane applications operating on the same control plane. An example of network services is IPTV, which can be produced by combining a unicast routing application to handle control traffic with a multicast routing application [CR17; Ble+14] to handle the media streaming traffic.

Control plane applications do not have to be designed for a specific SDN protocol but are often programmed against a specific controller API. Nevertheless, the programming model of the available SDN protocols influences the available functions and design styles for applications.

3 RELATED WORK

Control planes and controllers have been the subject of research since the introduction of SDN. We give an overview of the literature on control path bottlenecks and the state-of-the-art in their mitigation in Section 3.1. We investigate the state-of-the-art in network function chaining and resource efficient multicasting in Section 3.2. Finally, we present the identified research gaps in Section 3.3.

3.1 SOFTWARE-DEFINED NETWORKING DATA PLANE CHARACTERISTICS

The understanding of data planes, performance of the control path, and bottlenecks in literature are discussed in this section. The performance bottlenecks are categorized in anticipation of our own analysis provided in Chapter 4. This approach is taken to clarify the gaps in the existing literature. To understand this categorization, the notion of resources needs to be introduced. We use the notion of resources to describe components of the data plane element hardware that potentially can cause a bottleneck that affects either the control or the data path. Section 3.1.1 provides an overview of the topic of virtualization and isolation of control plane applications. An overview of literature on performance bottlenecks on the data path is given in Section 3.1.2. Existing strategies on how to mitigate performance issue such as limited flow table space are discussed in Section 3.1.3. Finally, we analyze the SDN devices that have been investigated in literature to provide an understanding of how to assess the published results and understand the main influencing factors in Section 3.1.4.

3.1.1 *Virtualization and Isolation*

Serveral surveys on SDN discuss the issue of virtualization. The survey on SDN virtualization by Blenk et al. [Ble+16c] provides a good overview of the issue. The topic of data plane control path resources is discussed, but as this survey will show, no attempt has been made in the literature to ensure all resources on data plane devices are discovered and isolated. Therefore, we will focus on papers that were published after 2015 when the first version of the paper was published [Ble+15c]. However, we still discuss the most important papers published before 2015, even if they are included in the survey of Blenk et al.

The need for virtualization is clearly stated in the SDN architecture [ONF14c]. However, no details on the specific requirements for virtualization are provided. Furthermore, no distinction is made between the virtualization of the control path and the data path of data plane elements. The ONF architecture proposes to conduct virtualization either on the data plane elements or in the SDN controller as denoted by ③ and ① in Figure 3.1. In addition to that, literature introduced the notion of SDN hypervisors [She+10]. Hypervisors are located between the data plane and the control plane, denoted by ② in the depiction, to provide data plane virtualization. However, hypervisors propose to operate multiple independent control planes on the same network, which is not applicable to ISP networks that require reliability and simplicity.

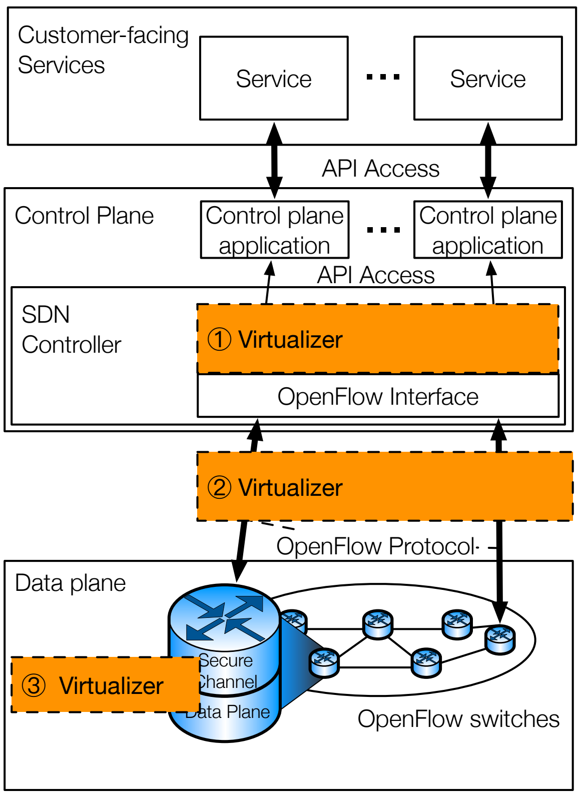


Figure 3.1: Potential locations of virtualizers.

Sherwood et al. [She+09] where one of the first to recognize the importance of control path bottlenecks for virtualization when proposing FlowVisor. They find the management system central processing unit (CPU) to be an important bottleneck and derive the processes that run on it and are affected by performance bottlenecks. Furthermore, they acknowledge the need to expose these bottlenecks through the SDN mechanisms. However, they do not attempt to provide a complete list of potential performance bottlenecks in the data plane and refer to solving the issues in future work. In conclusion, this paper, albeit being already published in 2009, discusses most of the relevant topics in virtualization.

Sköldström and Yedavalli [SY12] investigate the design space for placing virtualizers in the SDN architecture. They conclude that the management system of data plane elements is the best place to do so. The approach has many advantages including the possibility of full isolation of the CPU between tenants. The disadvantage is, however, that while this approach is expected to work well with few tenants, it might be too complex for hundreds of control plane applications. Furthermore, it requires the SDN control plane to control the hypervisors on the data plane elements in addition to their other tasks.

Dixit et al. [DKE14] propose a hypervisor called FlowBricks to enable multiple SDN controllers to access the same data plane. The assumption is that a single controller architecture is not able to provide all required network services. However, the virtualization approach focusses purely on logical isolation by translating the OpenFlow message contents. The order of the messages and their rate are not considered.

Mogul et al. propose their Corybantic framework [Mog+13] to handle conflicting policies of independent control plane applications. The goal is to find a configuration for a data center network that yields the highest revenue for the data center operator. By introducing a virtual currency that reflects the revenue contribution of each control plane application, a central coordinator selects the network configuration with the highest revenue. The paper relies on an ad-hoc data plane resource list and does not investigate control path resource dynamics. Furthermore, the resource demands are assumed to be well known in advance, which, e.g., does not reflect routing updates in unicast routing or demand spikes for multicast services.

Soulé et al. propose a data path resource-aware language for the northbound API called Merlin [Sou+14] that control plane applications use to communicate with the controller. In contrast to earlier works, Merlin enables control plane applications to specify not only the packet processing, but also the path a packet takes and, most importantly, data rate guarantees. Merlin achieves this by introducing a language with which traffic flows are selected, a path through the network is assigned including waypoints, and a data rate specification is given. However, again, ad-hoc data path resources are discussed, the issue of control path virtualization is not investigated.

Shin et al. [Shi+14] propose their approach, called Rosemary, to completely isolate control plane applications. The focus of the work is on the controller. Nevertheless, the authors acknowledge that all resources used by an application need to be monitored. However, the authors do not discuss control path resources. Blenk et al. [BBK15] propose to implement the SDN control plane itself using NFV and function chaining to enforce resource isolation between applications. They identify the data plane CPUs as bottlenecks and employ rate limiting to prevent overloading them. However, the impact of this rate limitation on control plane applications with different priorities is not investigated. Furthermore, additional control path resources are not identified or isolated.

Table 3.1: Control path virtualization features of SDN controllers compared to the bottleneck analysis conducted in this thesis.

Paper	**Goal**	**Location**	**CPU**	**Management NIC**	**PCI-e link**	**PCI-e controller**	**Flow table space**	**Flow table memory interface**	**Group table space**	**Group table memory interface**	**Meter table space**	**Meter table memory interface**	**Statistics counter table memory interface**	**Pipeline packet processing**	**Packet output to switch controller port**
FlowVisor [She+09; She+10]	Virtual-ization	①	⚠				V								
Corybantic [Mog+13]	Virtual-ization	①												VP	
Data plane virtualization [SY12; SJ13]	Virtual-ization	③	V												
FlowBricks [DKE14]	Virtual-ization	②													
Rosemary [Shi+14]	Virtual-ization	①													
Merlin [Sou+14]	Virtual-ization	①												VP	
CoVisor [Jin+15]	Virtual-ization	①					V	⚠							
Hyperflex [BBK15]	Virtual-ization	②	V												
LegoSDN [CTB16]	Virtual-ization	①													
SDNShield [Wen+16b]	Virtual-ization	①													
IVC/IVS [SPA16]	Virtual-ization	①③	V												
Onix [Kop+10]	Perfor-mance	①													
ONOS [Ber+14]	Perfor-mance	①													
OpenDaylight [Med+14]	Perfor-mance	①													

The predominant SDN controllers used in the industry today are ONOS and Open-Daylight. ONOS was conceived by Berde et al. [Ber+14]. The term resource appears in the context of resources, but no details are provided. The isolation of resources is mentioned in the future work section. Bottlenecks are discussed in the context of the controller only. In the presentation of OpenDaylight by Medved et al. [Med+14] the word resource appears two times in the text, but not in a relevant context. Therefore, we conclude that both designs strive primarily for performance. This approach makes sense for the goal of gaining acceptance in the industry but makes these designs susceptible to resource contention.

The papers analyzed in this investigation, the location where they suggest performing virtualization, and resources they identified and virtualized are listed in Table 3.1. The meaning of the symbols in the table that signify the handling of a hardware bottleneck is as follows:

- Identified only: ⚠
- Identified and virtualized: V
- Identified and virtualized with prioritization: VP

An overview of the locations where virtualization is suggested to be performed is depicted in Figure 3.1. The table is listing the related work references to this depiction by denoting the number of the virtualization location in the „location“ column.

3.1.2 *Data Plane Performance Bottlenecks*

In this section, the performance bottlenecks discovered in literature are discussed. We focus on approaches that identify bottlenecks and provide approaches on how to mitigate them. The most important papers on the topic of performance bottlenecks are discussed first, followed by an overview of all literature investigated.

The issue of consistent network updates, on which, e.g., Förster et al. provide a survey [FSV16], is related but not the focus of this investigation. If the consistency of network updates is prevented by resource contention on the control path, our investigation will support this use case as well. For ensuring that network-wide policies are activated consistently, the Time4SDN approach, proposed by Mizrahi and Moses [MM16] was recently integrated into OpenFlow and provides a solution for this issue in time synchronized networks.

Most SDN protocols and accompanying configuration protocols are designed to specify and reason primarily about functional aspects of data planes. This approach is reflected by the OpenFlow data plane model that covers functional aspects of matcher and actions, but not their non-functional aspects. Other aspects, such as the resource consumption and the resource-sharing behavior are often not covered, or, in the example of OpenFlow, introduced as an afterthought. One example is the flow table vacancy event feature that was introduced as late as OpenFlow version 1.4.0 [ONF13], which was released in 2013, four years after the release of OpenFlow 1.0 [ONF09]. Other control path aspects recent version of the OpenFlow protocol, as well as the OF-CONFIG protocol [ONF14a], report on are the size of flow tables, group tables, and meter tables. Another control path bottleneck that is acknowledged is sending packets from the ASIC to the control plane. The proposed solution to this issue is discussed in Section 3.1.3.

Costa et al. [Cos+17] propose a systematic approach to investigate the performance of SDN data panes. Unfortunately, the approach does not explain how the list of performance tests was derived and why. This leads to the assumption that, again, an ad-hoc approach is used to determine potential performance bottlenecks. Still, the authors investigate the behavior of the tested devices when their flow tables are full and discover that many of them exhibit unexpected behavior in this case. They, therefore, suggest filling flow tables at maximum to 90% of their capacity–which is lower than advertised by the corresponding OpenFlow primitives in many cases.

Rotsos et al. [Rot+12] provide in-depth measurements of three hardware and one software OpenFlow switch. They use a glass box approach and provide insights into the data path performance as well as on the control path. They investigate the performance of different OpenFlow message as well as their interaction. To do not, however, try to systematically create a performance model of the data plane.

Lazaris et al. [Laz+14] argue that the diversity in performance characteristics of SDN devices is not captured adequately by existing SDN protocols, e.g., OpenFlow. Therefore, control planes need detailed information on the expected control path performance of the devices. To that end, the authors present an inference system that sends OpenFlow message patterns and measure the response in the data as well as the control path. They describe the unpredictable behavior of TCAM table sizes because, on some switches, some combinations of packet match fields yield dramatically different maximum number of flow entries in a table. Furthermore, they investigate the flow_mod message in detail and find that the priority, as well as the order and number of existing entries in a table, impact the performance of adding entries. The performance of modifying entries, however, is mostly constant. They assume that the approach of some vendors to use the management system as a slow path for packets. However, this approach cannot be considered viable for high-performance networks and multi-gigabit traffic [KMH14]. The hardware performance interference patterns seem to be derived from an ad-hoc model of OpenFlow switches. The devices are investigated as black boxes, and there is no discussion of how the specific characteristics where chosen. Furthermore, the question whether all relevant configurations and combinations of parameters are investigated is not discussed.

The authors then present an approach to analyze application requests to schedule them for best performance. The analysis process is designed to ensure that the dependencies between requests are kept even after the optimization process. The results are very promising and show that the approach works well for the investigated use cases.

He et al. [He+15] argue that the control path latency is crucial for many services. They dissect the components that make up the latency but do not recognize virtualization as a factor. They describe the hard- and software of OpenFlow switches and investigate two use cases: packet forwarding along the control path and flow table updates–a reactive flow installation scenario. While the process described is detailed and includes all relevant components, it is specific to a configuration and cannot be generalized as described. The issue of virtualization and its effects is not discussed.

The literature on security issues in SDN focus on specific issues, but again, work with ad-hoc models of the data plane [SNS16; Yoo+] and do not provide additional insights into the issue.

Table 3.2: Performance bottlenecks in literature compared to the analysis conducted in this thesis.

Location	**Resource**	**Identified**	**Optimized**	**Virtual-ized**
Manage-ment system	CPU	[CB17a; Laz+14; BR13; Nar+12; KHK13; Cur+11; Amb+17; Wan+14; Rot+12; SY12; Cos+17]	[BR13; Nar+12; KHK13; Cur+11; Wan+14; SY12]	
Manage-ment system	Management NIC			
Manage-ment system	PCI-e link			
ASIC	PCI-e controller			
ASIC	Flow table space	OpenFlow [ONF15], [Qia+16; Nar+12; Yu+10; Cur+11; Guo+17; Yoo+; KHK13]	[Qia+16; Yu+10; Cur+11; Guo+17; KHK13]	
ASIC	Flow table memory interface	[CB17a; Laz+14; Qia+16; HYS; Kat+16; Jin+14; Rot+12; Wen+16a; Ngu+18]	[CB17a; Laz+14; Qia+16; Kat+16; Wen+16a]	
ASIC	Group table space	OpenFlow [ONF15]		
ASIC	Group table memory interface			
ASIC	Meter table space	OpenFlow [ONF15]		
ASIC	Meter table memory interface			
ASIC	Statistics counter table memory interface	[Cur+11; Rot+12]		
ASIC	Data Path	[Jar+11; Rot+12]		
ASIC	Packet output to switch controller port	OpenFlow [ONF15], [He+15; Nar+12; Bas+17; Amb+17; Wan+14]	OpenFlow [ONF15],[Nar+12]	Open-Flow [ONF15]

Chen et al. [CB17b] argue that for certain applications (traffic engineering, mobile networks, cyber-physical systems), control path performance guarantees are required. Based on the observations made in [Kuz+18; He+15; Laz+14] they conclude that the primary source of unpredictability is the number of flow entries in a table. To mitigate this effect, they use two TCAMs or TCAM partitions, one as the main table and one as insertion cache. The number of entries in the cache is kept small to achieve guarantees for inserting entries there. Then, the entries are migrated to the main table to keep the cache table size small. The approach is evaluated using a simulator and shows promising results. The approach aims to improve the design of OpenFlow switches.

However, the effect of virtualization is not discussed. Guarantees can only be given if the system has enough time to copy the entries into the main table. How this approach affects the SDN protocol is not discussed, neither is how the SDN controller knows of these guarantees.

The complete overview of the literature investigated, and the resource discovered there is listed in Table 3.2. The table clearly shows that fixed properties like table sizes are well investigated as well as the dynamics of the management system and table updates. However, we found no papers discussing topic prioritization on the control path, virtualization, or its effects on control plane applications.

3.1.3 *Control Path Bottleneck Mitigation Strategies*

Performance mitigation strategies are included in the tables of the preceding two sections. There are four control path resources for which mitigation strategies exist: the management system CPU, flow table space, the flow table memory interface, and the packet output from the data plane to the controller.

The approaches optimizing the flow table space follow mostly two approaches that are implemented on the controller without the knowledge of applications: merging multiple entries and distributing entries on multiple switches. The fist approach shows promising results, however, merging entries leads to a reduced visibility for applications, because the counters that belong to an entry will be lost. Therefore, the question remains how the applications signal that they accept the potential loss of counters. Furthermore, if the interface between the controller and the applications is at least as powerful as the OpenFlow protocol, the sheer amount of options per entry such as priority, timeouts and counter push events make it questionable if this approach is realistic for large scale deployments. The second approach, as the first one, requires the applications to allow the controller to significantly modify their decisions. This requires applications to be able to cope with this, e.g. if they expect flow counters to operate on a specific location. Again, the question remains how applications should signal that such kind of modifications are acceptable for their use case.

The proposals for increasing the throughput of the flow table memory rely on specific properties of the TCAM. One example is the usage of caching, e.g. by installing flow entries in an empty TCAM table first, before moving them to the main table [CB17b]. This approach reduces the flow entry addition time in the experiments, since the addition time was shown to depend on the number of existing entries in a table. While these approaches are very promising and should be used where possible, they still can lead to overloaded flow table memory interfaces. In the case of the described example the cache table can overflow when a large number of flow entries is added. For this situation none of the papers provides a mitigation.

Sending packets from the ASIC to the control plane is a well-known issue that is described in the OpenFlow specification [ONF15]. The authors propose to apply rate limiters to traffic before sending it to the control plane. This approach can prevent performance bottlenecks and if multiple rate limiters are combined with prioritization even provide full virtualization of the resource. Unfortunately, many of the available OpenFlow devices are not able to apply multiple meters before sending a packet to the control plane. Nevertheless, we expect this approach to be used in future, more capable devices e.g. P4 programmable ASICs as proposed by Bosshart et al. [Bos+13].

Mitigations to prevent the management system CPU from becoming overloaded propose to reduce the number of messages sent by the controller. One approach proposed in to apply a rate limiter e.g. by Blenk at al [BBK15]. The approach is well suited to prevent overload, however, it requires an exact knowledge of the number of SDN protocol messages that the CPU can process. This number is difficult to come by, because it cannot be expected that all messages need the same amount of processing resources. Furthermore, there are other processes running on the CPU that cannot be controlled by the control plane, e.g., packets sent from the data plane to the control plane as described before. One approach that completely mitigates this issue is proposed by Sköldström and Yedavalli [SY12]. It uses operating system (OS)-level virtualization on the data plane's management system to isolate multiple OpenFlow agent instances. This approach provides complete virtualization of the management system. However, it also requires a complete OpenFlow agent instance per tenant. Therefore, this approach is suitable for isolating tenants, but might be difficult for a large number of control plane applications.

3.1.4 *OpenFlow Data Plane Devices in Literature*

To better understand the results provided in the literature on the performance characterization of data plane elements, we surveyed the used OpenFlow switches in academia. The result of the survey is listed in Table 3.3.

Table 3.3: Overview of investigated OpenFlow switches in academia.

Model	ASIC/NPU	CPU	OS	Switch softw.	Literature
Arista 7050 [a]	Broadcom Trident+	AMD Turion II Neo N41H	Arista EOS (Linux)	propr.	[Jin+14]
Dell 8132F[b]	Broadcom Trident +	unknown	propr.	propr.	[Kuz+18]
EdgeCore AS5712-54X[c]	Broadcom Trident 2	Intel Atom C2538	PicOS (Linux)	PicOS	[CB17a]
HP 2920[d]	HP ProVision	Tri Core ARM1176	propr.	propr.	[Cos+17]
HP 5400 zl[e]	HP ProVision	NXP/Freescale MPC8540	propr.	propr.	[Kuz+18; Cur+11; She+09]
HP ProCurve 6600 (J9451A)[f]	HP ProVision	NXP/Freescale MPC8540	propr.	propr.	[HYS; Wan+14]
IBM G8264	Broadcom Trident 2	unknown	propr.	un-known	[He+15]
Intel Fulcrum Switch	Intel FM6000 series	unknown	propr.	un-known	[HYS; He+15]
NEC IP8800	unknown	unknown	propr.	un-known	[She+09]
NEC PF5240[g]	Broadcom unknown	PowerPC	propr. (NetBSD)	propr.	[Ngu+18; DK15; KMH14]
NEC PF5820[h]	Broadcom Trident+	unknown	propr.	propr.	[Ber+14]
NoviFlow NoviSwitch 1132[i]	EzChip NP4	Intel Core i7-620LE	NoviWare (Linux)	Novi-Ware	[Kuz+18]
Pica8 P-3290[j]	Broadcom Firebolt 3	NXP/Freescale MPC8541E	PicOS (Linux)	PicOS	[BR13; Kuz+18; Ngu+18; Jar+11; Kat+16; DK15]
Pica8 P-3297[k]	Broadcom Triumph 2	NXP/FreeScale P2020	PicOS (Linux)	PicOS	[Cos+17]
Pica8 P-3780[l]	Broadcom Firebolt 3	NXP/Freescale MPC8548E	PicOS (Linux)	PicOS	[Wan+14]
Quanta LB4G	Broadcom 56514	unknown	propr.	un-known	[HYS]

[a] Arista. 7050 Series 10/40G Data Center Switches Datasheet. Apr. 2017.
[b] Dell. Dell PowerConnect 8100 Series Datasheet. SS806_Dell_PowerConnect_8100_Se- ries_2012-12-31. Dec. 2012.
[c] Edge-coreE. AS5712-54X 10GbE Data Center Switch Datasheet. Data sheet EC-DS-0118-07.
[d] HP. HP 2920 Switch Series Datasheet. 4AA4-5213ENN. Feb. 2015.
[e] HP. HP ProCurve Switch 5400zl Series Datasheet. 4AA2-6511ENW. Apr. 2010.
[f] HP. HP ProCurve 6600 Switch Series Datasheet. 4AA2-3898ENW. May 2009.
[g] NEC. NEC ProgrammableFlow UNIVERGE PF5820 Datasheet.
[h] NEC. NEC ProgrammableFlow UNIVERGE PF5240 Datasheet.
[i] NoviFlow. NoviSwitch 1132 High Performance OpenFlow Switch Datasheet. DS2017- NS1132-400-03. 2013.
[j] Pica8. Pica8 P-3290 Datasheet.
[k] Pica8. Pica8 P-3297 Datasheet.
[l] Pica8. Pica8 P-3780 Datasheet.

Before describing the devices and their differences, we give a brief overview of the relevant architectural components. The data plane elements consist of, an often proprietary, packet-processing ASIC that is connected to a standard management system, often based on the widely used x86 architecture. The interconnection between CPU and ASIC often relies on PCI Express (PCIe). On the management system an OS, e.g., the Linux OS, provides an abstraction layer for the OpenFlow agent to operate on. The interface to the ASIC is provided by a driver that runs on the Linux OS but is often proprietary, closed source, with no publicly available documentation. The management system itself has a dedicated network interface that is connected to a management network. This network is used to connect to the OpenFlow controller. Communication from and to the OpenFlow controller is processed by the Linux OS first, then handed to the OpenFlow agent. The agent translates the OpenFlow commands to API calls of the ASIC driver. In turn, the driver translates the API calls to messages that are sent over the PCIe bus to the ASIC. There, a processing unit handles the packets and initiates the requested changes in the ASIC. However, it becomes clear that the performance of the management system and the specific implementation of the OpenFlow agent are crucial factors when assessing the performance of control paths on data plane elements.

Interestingly, none of the devices investigated in literature can be classified as designed for ISP core networks. To the best knowledge of the authors, the availability of OpenFlow in high-end, high-throughput devices is limited. Furthermore, information on these devices is not always available to the public. We assume this is because of their limited market being large ISPs, to which information are provided directly, instead of being released to the public.

The first observation is that the devices from Pica8 and HP, followed by NEC are most often used in academia, as indicated by the number of papers that investigated this switch in the „Literature“ column. The only device that uses an NPU for packet processing instead of an ASIC like the rest is the NoviFlow NoviSwitch 1132. The results on the flow entry modification performance of such a device by Kuzniar et al. [Kuz+18] is an order of magnitude better than the other, ASIC based devices.

Many of the devices that have been used in academia have been discontinued by their vendors such as the Fulcrum Monaco, the NEC IP8800, the Arista 7050, and Quanta LB4G devices. In addition to that, we do not know the management system's specifications, which makes it difficult to assess the significance of the results.

Of the remaining devices, the ASICs used by the devices are mostly built by Broadcom, except for the devices from HP. The HP devices listed in the table all rely on the HP ProVision ASIC. Unfortunately, the depth of OpenFlow support by this ASIC is lacking. In contrast to the Broadcom-based devices not even all the basic match fields are supported[1]. Hence, it cannot be considered state-of-the-art OpenFlow implementation. Furthermore, the performance of the management system CPU is severely restricted. This means that the while the research conducted on the OpenFlow implementation of this line of HP devices gives in impression of the spectrum of available devices, the results cannot be considered as representative, especially when it comes to its control path performance.

1 HP. *HP Switch Software OpenFlow v1.3 Administrator Guide K/KA/KB/WB 15.18*. 5998-8148b. Apr. 2016.

In general, we find a high proportion of the devices rely on Broadcom ASICs and run the Linux operating systems. The fact that there are still differences in the reported performance characteristics of the SDN primitives that involve the same ASICs support the finding that the SDN agent software has a significant influence on the control path performance.

We conclude our discussion of data plane elements by stating our main findings:

- Control path performance results depend on the management system and the packet processing ASIC.
- Many results available in academia cannot be considered up to date anymore.
- The packet processing hardware architecture has a significant influence on the control path performance.
- The performance results in the literature show a significant heterogeneity.

3.2 CONTROL PLANE APPLICATION EFFICIENCY AND BOTTLENECK MITIGATION

Two example control plane application are discussed in this thesis and are adapted to handle control path performance bottlenecks. The literature on reacting to control path bottlenecks has been discussed in the preceding section. In this section, we present the literature on the two applications themselves. The existing research on network function chaining is presented in Section 3.2.1 followed by the research on SDM in Section 3.2.2.

3.2.1 *Network Function Chaining*

In network function chaining, also termed network function chaining in literature, network traffic is arbitrarily moved through a chain of VNFs operating on x86 servers in a data center. The network interconnect between the servers are hardware switches, the VNFs are interconnected to the hardware switch through a software switch running on the same server.

StEERING, proposed by Zhang et al. [Zha+13] was one of the first approaches to optimize network function chaining specifically for OpenFlow. SIMPLE, introduced by [Qaz+13], uses a different approach by creating tunnels between the server nodes where the VNFs are located. Both approaches are designed to work with VNFs that do not have to be specifically adapted to be included in a function chain. Both approaches introduce valuable optimizations for their specific use case. Furthermore, both reveal a similar flow entry update bottlenecks, which is investigated in our contribution. However, the approaches are designed for a specific use case and, more importantly, they do not investigate the issue of control path bottlenecks in case of spikes in entry addition events. Network Service Header, specified by Quinn et al. at the Internet Engineering Task Force

(IETF) [RFC8300] takes a different approach that requires the VNFs to be adapted for this specific protocol and therefore has slightly different flow update characteristics.

While not a function chaining approach, CacheFlow, proposed by Katta et al. [Kat+16] introduces the concept of load-sharing of flow entry updates. In CacheFlow, the flow update load is shifted from a hardware switch to a co-located software switch to mitigate a flow update bottleneck and the limited flow table space. The hardware switch is used for a few flows that require high throughput, while the software switch is used for many low throughput flows. In contrast to that, we propose to mitigate a flow update bottleneck on a software switch by sharing its load with an upstream hardware switch.

3.2.2 *Software-Defined Multicast*

Multicast is a network service that could be useful for many applications on the Internet–if it was globally available. IP multicast [DC90] was not able to achieve this [Dio+00], which is why SDM was conceived by Rückert and Blendin et al. [RBH15] specifically to be used in ISP networks. SDM uses multicast forwarding where possible to increase the transmission efficiency and unicast forwarding where required to reach all parts of the network. The concept itself, as well as its integration into other network services and the global Internet architecture, is exhaustively discussed in the dissertation of Rückert [Rüc16] and related papers [RBH13; Rüc+14; Rüc+15; Rüc+16].

We focus on the state efficiency of multicasting, which has not been investigated by Rückert in his work. Therefore, the following overview of the related work also focusses on the network state efficiency.

In traditional IP multicast, each router along the path of a multicast packet must have a corresponding multicast entry in its forwarding table. Traditional routers often have a specific part of their hardware forwarding table reserved for multicast forwarding. This approach, its requirement for an entry on each router in the path as well as the strict limitation for forwarding table entries, limits the number of multicast groups in a network domain. This bottleneck is exacerbated by the fact that Internet content typically follows a Zipf distribution [AH02], which has been confirmed for streaming services [SMZ04]. A set of Zipf distributed multicast groups has very few very large groups with many members and a very large number of small groups. Specifically, this large number of small groups limits the usefulness of IP multicast.

Boudani et al. [BC02], as well as Apostolopoulos and Ciurea [AC05], propose a mitigation approach for this by merging overlapping parts of multiple multicast trees. Thereby, when many multicast trees share the same set of recipients the amount of multicast forwarding entries can be reduced significantly. However, given the Zipf-distributed characteristics of multicast groups, the number of overlapping trees is not expected to be large since most trees have few members only.

Leaky aggregation relies on the same observation of overlapping multicast trees but forgoes the correctness requirement. If two multicast trees overlap mostly, but not exactly, they are still merged. This approach has the advantage that the potential for merging trees increases but has the drawback that some recipients will receive unwanted multicast traffic. Thereby, the transmission efficiency is reduced. This concept is investigated in detail by Radoslavov et al. [REG99] and provides the motivation for our ASDM approach.

Stoica et al. [SNZ00] propose REUNITE, in which multicast is implemented by using unicast addresses to create multicast behavior through recursive unicast trees. Unicast trees are located on a router and identified by the IP address of the router and the destination port number. A unicast tree is the combination of this identified and a list of outgoing packet duplicates and their next unicast tree identifier. Using this recursive scheme, the packets are sent to a unicast tree, where they are duplicated, and the destination IPs and port of the respective next unicast tree is written into each packet. REUNITE saves forwarding table entries because unicast tree state is only kept at routers that perform packet duplication. Routers that are not involved are skipped using unicast addressing between the duplication nodes that host unicast trees. However, REUNITE does not use the group size to select the tree structure, which still leads to a higher forwarding state consumption that necessary with many small multicast groups. Still, the approach is promising and is used in our ASDM design.

Xcast, proposed by Boivie et al. [RFC5058] uses a different approach to multicast addressing. Instead of keeping the multicast state in the network, it is kept in each packet header in the form of a list of destination IP addresses. Each node parses the Xcast header, compares it to its unicast forwarding table and creates duplicates for the packet if it finds different next hops for the IP addresses in the destination list. The Xcast approach removes the need for keeping the multicast state in the network, one of the primary reasons why multicast is not widely deployed today. However, this approach has the drawback that the Xcast packet header introduces considerable transmission overhead. This means that the approach is only suitable for small groups, since for large groups with, e.g., 500 members the header space required to store the destination addresses is with 2000 bytes already larger than the standard maximum packet size.

An improvement to Xcast was first proposed by Blendin [Ble13] is explicit multicast with bit indexes. Instead of using global IP addresses in the packet header, each potential destination in a network domain is assigned a numerical ID. Each packet includes a one hot encoded bit string whose bits each signify a different destination address. Thereby the large overhead of Xcast is reduced. The explicit multicast approach was later independently introduced and further developed by Wijnends et al. in the IETF to the Bit Indexed Explicit Replication (BIER) protocol [RFC8279]. Giorgetti et al. provided the first BIER implementation using SDN, specifically OpenFlow [Gio+17]. However, to the best knowledge of the author, the source code of the implementation is not publicly available. Furthermore, there is no study on the network state memory efficiency of BIER. None of the described approaches is prepared to handle forwarding state bottlenecks in the data plane.

3.3 DISCUSSION AND RESEARCH GAPS

We compared the understanding of bottlenecks in the control path of the SDN date plane to the results of our discovery process that will be presented in Chapter 4. Thereby the gaps in the literature that are caused by using the ad-hoc methods to investigate potential bottlenecks in data plane devices are shown. The work in academia on virtualizing data path as well as logical aspects of flow entries are impressive. Furthermore, the work on isolating applications on the controller platform appears to have solved virtually all issues in this area. Unfortunately, the control path of the data plane has been left out of most of these investigations. Specifically, bottlenecks in the throughput aspects of the control path of data plane elements have been neglected. Failing to account for this kind of performance issues can lead to uncontrolled behavior of control plane applications. The importance of complete performance control is demonstrated by Basta et al. [Bas+17] on the instance of overloaded SDN hypervisors. The authors demonstrate that hypervisors can introduce additional latency and therefore impair the ability of control plane application to interact with the data plane without the knowledge of the SDN control plane.

The discussed bottleneck mitigation approaches advance the state-of-the-art significantly. Still, the fundamental issues are not solved, which is how to discover, control, signal and handle inevitable performance bottlenecks on the SDN controller. One part of this field is the issue of control path prioritization between applications. Although it is critical for ISP networks, is not addressed in the existing literature.

Furthermore, the handling of control path bottlenecks in applications has not been investigated in depth yet. This is true for network function chaining applications as well as for multicasting applications. We, therefore, conclude that an approach to tackling control path bottlenecks from the perspective of the SDN controller is required, as well as bottleneck mitigation approaches for control plane application.

4 VIRTUALIZING THE CONTROL PATH OF SOFTWARE-DEFINED NETWORKING DATA PLANES

In this chapter, we focus on Research Questions 1.1 and 1.2 that are stated in Chapter 1. Investigating the existing literature on performance characteristics of control plane applications we identified a research gap in the control path performance management in SDN environments. Specifically, the literature is lacking a systematic approach on how to identify potential performance bottlenecks on the SDN control path and control them through virtualization. To that end, we introduce a systematic approach to analyze SDN data planes, identify potential control path performance bottlenecks, and introduce adequate virtualizers on the SDN controller to prevent bottlenecks from becoming uncontrolled interference between control plane applications.

We argue that overloaded hardware components, termed resources cause performance bottlenecks. Potential bottleneck components are identified by investigating the SDN data plane for hardware components that are affected by control plane application and in turn, can affect the performance of other control plane application. For this analysis, we introduce the notion of a resource topology to analyze dependencies between resources and their interaction with SDN protocol primitives.

The context and goals for this approach are provided in Section 4.1. In Section 4.2 our approach to discovering and virtualizing resources is presented. We discuss selected contemporary data plane devices before creating a representative resource topology for a state-of-the-art SDN data plane device in Section 4.3. We identify throughput bottlenecks in the control path as a new class of bottlenecks for which the literature has not presented a virtualization approach yet. Therefore, we present a virtualization approach for the representative throughput bottleneck of packet match table updates in Section 4.4. Finally, we discuss our approach, its advantages, and disadvantages in Section 4.5.

The investigation in this chapter partially relies on input from two papers [BH14; Ble+16a] and four supervised student theses by M. Härdtlein, S. Bleidner, F. Villa-Arenas, and X. Zhang [Här17; Ble15; Vil18; Zha18].

4.1 ASSUMPTIONS AND REQUIREMENTS

Networking in general, and SDN specifically, is a wide field with diverse architectures. Our understanding and assumptions for ISPs and the services they offer are presented in Section 4.1.1. The SDN architecture we work with and the requirements for our approach are described in Section 4.1.2.

4.1.1 *Internet Service Provider Networks and Network Services*

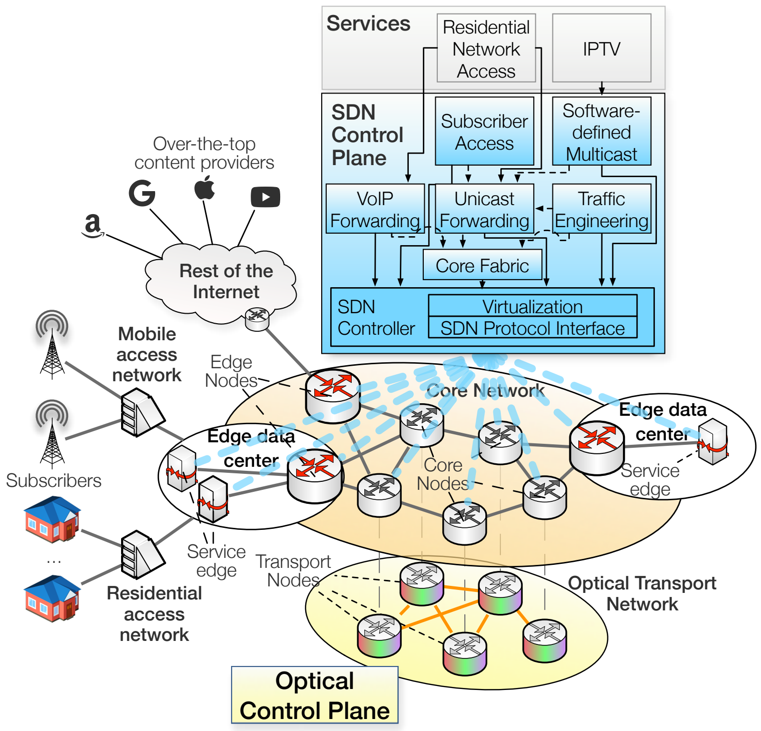


Figure 4.1: A technological view on ISP networks as used in this thesis.

A technical overview of ISP networks is given in Figure 4.1. The core network is made up to two parts: an optical transport network that provides optical network links and an IP network using these links for interconnection. This is important, because in contrast to the core network, in edge and data center networks the network devices operate their links directly, without the help of an additional optical control plane. This means that the capabilities of a link are determined by its endpoints, which are SDN devices.

Optical transport networks, although they often could be called software-defined for their use of remotely configurable equipment such as ROADMs, are unlikely to adopt a standard SDN protocol. This is because their equipment is much closer to the latest research. Therefore, the technology is often proprietary and not standardized, which prevents the application of a standardized SDN protocol. Hence, this part of ISP networks is therefore not included in our investigation.

The network edge is assumed to be organized as small data centers, e.g., as envisioned by Peterson et al. in their Central Office Re-architected as a Data Center (CORD) approach [Pet+16]. These CORD data centers are also expected to host the service edge, which is the customer-facing part of the network.

The conceptual overview of the relevant areas of ISP networks is given in Figure 4.2. We investigate the core and edge parts of the network, as well as data center networks. In these areas, network services are implemented and provided to the customers. Therefore, these are the areas where SDN is expected to provide the highest efficiency improvements. Access networks often use specialized network technology. This is because they have a different technological focus that the other parts of the network: providing connectivity at the lowest possible cost. However, the costs in this area of the network are dominated by traditional construction costs and government regulation. Network management and traffic steering are restricted to forward network traffic from subscribers to the closest edge data center. Therefore, it is not relevant for our focus on areas of ISP networks where SDN is expected to improve the management and control efficiency.

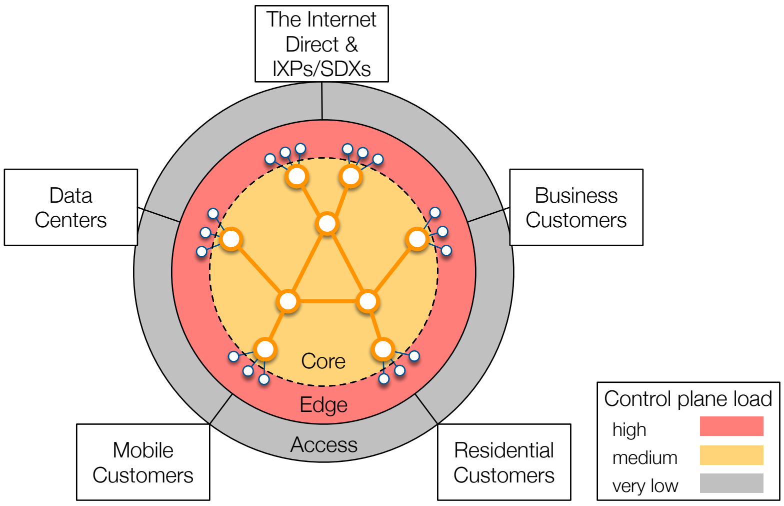


Figure 4.2: ISP network areas color coded by control plane load.

The network areas in Figure 4.2 are colored by their expected control plane load. Access networks are expected to be static, with connecting and disconnecting links of customers being the only events that cause control plane load. The elements of the core network, both packet switching and the optical transport elements, are expected to be modified in existing networks only in case of an element failure or for traffic engineering. Both events are not expected very often, and intervals between events are assumed to be in the magnitude of hours. Finally, the network edge is the area of the network where control plane applications are active to implement network services. Fundamental services like unicast connectivity are provided at the edge by translating IP destination to paths through the core network fabric. Customers are authenticated, and their contracts are enforced at the edge. We, therefore, expect considerable control plane load in this network area. Reasons for that might be inter-autonomous system (AS) routing updates, customer access events, or interactions of customers with network services. An example for the latter is when a customer switches a channel in multicast-based IPTV, which requires the network edge to update its multicast packet delivery configuration. In the remainder of this work, we will generalize mobile, residential, and business customers simply as customers. From a technical perspective, they represent networks that are connected to the ISP through the access area.

We observed two main characteristics for network services that are implemented by control plane applications and operated in ISP networks: control plane activity is expected to be low on average with sudden spikes of activity and an inherent need for prioritization of network services. The first observation is a natural outcome of the design approach for network infrastructure, which is to design for peak load instead of the average load. Peak load events for network services, e.g., can be the failure and subsequent re-activation of a device at the network edge. An example is the failure of a mobile network access node, which must re-activate thousands of customer connections on other devices as quickly as possible. In contrast to that, the average load of a mobile network access service is expected to consist of handovers for moving customers and few customers that switch their mobile devices on and off [Jin+13]. The same is true for the unicast routing service. This service is designed to handle the failure or connection event of a Border Gateway Protocol (BGP) router that provides routes from and to neighboring ASs. In such an event, a BGP router usually receives the whole Internet routing table with about 732,000 unique IPv4 prefixes[1]. Depending on the significance of the neighboring BGP router, such an event might result in up to the same amount of routing entry modifications in the data plane.

The need for prioritization of network services is caused by two effects: first, dependencies between services and second, commercial interests. Services such as the core network forwarding, or unicast routing are essential for all other services provided by an ISP as depicted in Figure 4.1. Therefore, these services should always take precedence over other, less critical services. The latter argument is already an example for commercial interests of the ISP. In cases where the control plane load is higher than the available processing capacities, the network operator has to decide which network services should continue to operate, and which should be impaired. In such cases, the ISP will let those network services be impaired that generate the smallest economic loss. For example, the core network is the foundation for many other services and should therefore never be impaired. Add-on services that generate only a small part of the ISP's revenue are better candidates to be starved of resources. Therefore, prioritizing network services and, subsequently, control plane applications are essential for efficient and reliable ISP network operations. Furthermore, these situations are examples where low priority applications need to be able to adapt to unexpected resource starvation on certain or all data plane elements. Finally, control plane applications can fail or contain errors. The controller must ensure that a failed or erroneous application cannot cause a failure of the controller or degrade the performance of other control plane applications.

1 Geoff Huston. *CIDR REPORT*. Accessed: 2018-8-27. Aug. 2018. url: https://www.cidr-report.org/as2.0/.

Another essential aspect of ISP and the closely related carrier networks is their operational complexity. Due to the importance of the network to the ISP companies, the general approach to the deployment of new technology is conservative [Lev+14]. Furthermore, to ensure the operating staff can maintain the system, the network management must be understandable for humans. One negative example is Generalized Multi-Protocol Label Switching (GMPLS) [RFC3945] which aimed at combining the widely used MPLS for packet networks with optical transport network features. Unfortunately, GMPLS ended up being an overly complex platform [Liu+12; Far10; DPM12] with no gradual update path from existing technology. Therefore, although GMPLS was developed for a long time and by many partners, it was not widely adopted by the carrier community [DPM12]. Hence, we to prevent our design to be deemed too complex, we require them to be as understandable and maintainable for their human operators as possible.

We summarize the information requirements for control plane application as follows: To achieve resource efficiency for applications, the controller must collect information on their resource consumption. To enable the controller to ensure reliable operations, in addition knowing all performance-relevant information, it must ensure that operating a control plane application does not have unexpected side effects. Furthermore, applications need to receive information on the performance of the control path to handle bottlenecks in a controlled manner. Finally, the entire system should be designed to be comprehensible by human operators to allow them to intervene in the case when the automatic reliability measures are not sufficient.

4.1.2 *Software-Defined Networking*

OpenFlow is one SDN protocol for controlling data planes and the de facto standard in industry and academia. OpenFlow is well-defined, used in the industry [Jai+13], and has been thoroughly investigated in academia [Kre+15]. There are few alternative protocols such as Forwarding and Control Element Separation (ForCES) [RFC5810], OVSDB [RFC7047], and Netlink [RFC3549]. However, to the best knowledge of the author, no generally available hardware devices exist for ForCES. OVSDB is available in addition to OpenFlow on hardware devices that use the Open vSwitch software as OpenFlow agent. It seems to be used less than OpenFlow, and existing academic literature focusses on OpenFlow instead. Netlink focusses on use cases where the controller and the ASIC are located on the same devices, which is not the scope of this work. Therefore, we study OpenFlow in this work as a representative for SDN protocols in general. We are confident that the findings presented in this thesis apply future SDN protocols as well.

OpenFlow version 1.3 is used in many applications and is a significant release since from there on, new features added in new releases are also available as extensions to OpenFlow 1.3. Some devices and controllers support OpenFlow 1.4, and even fewer the latest version 1.5. We assume OpenFlow 1.3 to be the baseline version and investigate features added in later versions as needed. While it is possible to operate the OpenFlow controller directly on data plane devices, we assume that the OpenFlow controller is operated on a dedicated server.

OpenFlow was initially proposed to be operated in two different modes: reactive and proactive. Reactive mode requires much interaction between the data plane and the controller for every new flow in the data plane. Flow table entries are installed on demand when a packet requiring them arrives. For this reason, it was found to be problematic and impeding the data plane performance. In proactive mode, flow entries are installed by the controller independently from the actual traffic on the data plane. We are convinced that this approach should be chosen over the reactive mode whenever possible. Therefore, we assume that the investigated SDN networks are operated in proactive mode.

We focus on SDN data plane elements that use ASICs for packet processing in this chapter. These are the most widely used data plane elements today, as discussed in Section 3.1.4 and provide high performance. Some of these data plane devices use a software switch on the device, termed the slow path, for caching flow entries or conducting packet processing that is not available in the ASIC[2]. However, this approach can severely impact the data path throughput of these devices [KMH14]. Therefore, in the context of high-throughput ISP networks, this approach should not be used. We assume in our investigation that packet forwarding happens on the data path in the ASIC of the switch only.

OpenFlow controllers are understood to be logically centralized. Logically centralized means that while they constitute a distributed system, their distributed nature is hidden from the data plane elements as well as from the control plane applications [Ber+14]. We accept this approach and will discuss OpenFlow controllers and control planes as if they were single instances, abstracting from their distributed nature. The mechanisms investigated do not depend on distributed features of the data plane.

We assume the usage of OpenFlow for the controller-to-application interaction as well as for the interaction between the controller and the data plane. We do this even though recent research by Schwabe et al. [SAK16] suggests that it may not be suitable at least for some of the corresponding tasks. The reason for this choice is that no widely deployed standard exists for control plane application APIs, termed the northbound interface, so OpenFlow is the only available choice.

We assume the virtualizer to be placed in the SDN controller. The effects of control path bottlenecks can be mitigated if the SDN controller can control the performance bottlenecks and their load and if this information is provided to the affected control plane applications. Therefore, we require the SDN controller and the control plane applications have enough knowledge to react to unexpected situations to prevent unpredictable network behavior as a result. Once the SDN controller notices performance issues, it should forward relevant information to the affected control plane application. From this approach follows that a controller must exclusively control all data plane devices

2 Pica8. *PicOS Open vSwitch Configuration Guide*. Version 1. Jan. 2017.

in a domain. Furthermore, because the controller requires full control over all data plane devices, virtualization features should not be placed on data plane devices. Our proposed view on the SDN architecture that is used as a basis for this work is depicted in Figure 4.3.

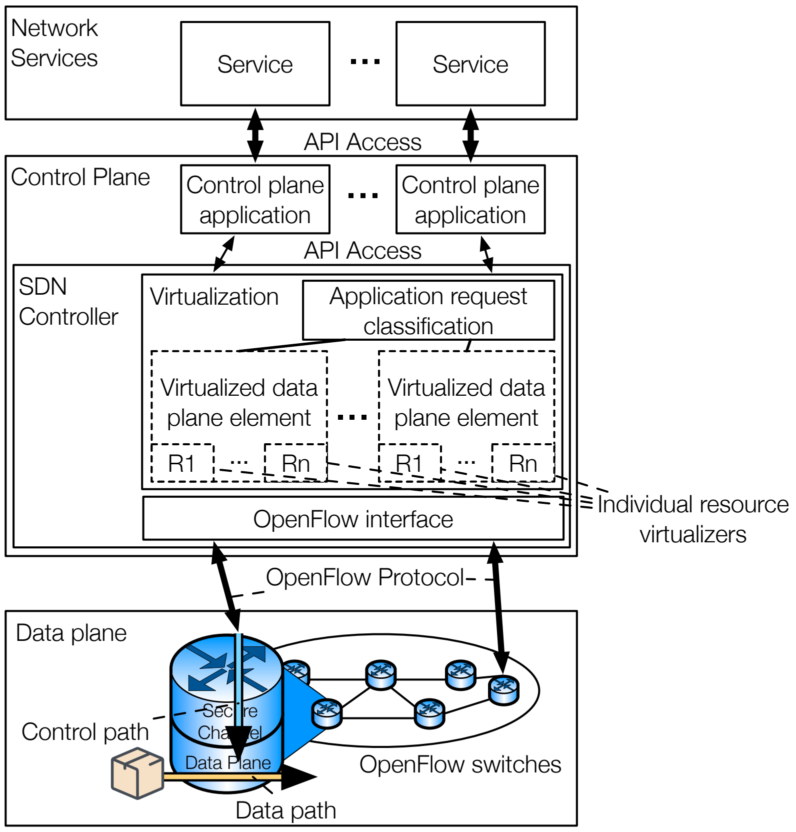


Figure 4.3: Overview of the SDN architecture as proposed in this thesis.

4.2 A RESOURCE-ORIENTED DATA PLANE VIRTUALIZATION APPROACH

The development of a new approach to analyzing SDN data path elements is required, because, to the best knowledge of the author, no systematic approach to virtualizing hardware data plane elements has been proposed yet.

An overview of the design of the approach is given in Section 4.2.1. The term resource and its use in the context of this work are defined and discussed in Section 4.2.2. Our approach to analyzing data plane devices to discover all relevant resources is introduced in Section 4.2.3. Finally, the design on how to integrate the discovered information into the SDN control plane and how to use it to create virtualizers for resources is described in Section 4.2.4.

4.2.1 *Overview*

The goal of this section is to give a high-level understanding of how our approach provides an answer to Research Question 1.1 and 1.2. While doing so, we will briefly introduce the relevant terms to improve readability. The detailed definitions of the terms will be introduced in the subsequent sections.

Our approach is motivated by the work of Gregg [Gre13] who introduced the utilization, saturation, errors (USE) method to investigate performance issues. Gregg suggests systematically investigating performance issues on servers through a bottom-up approach. Starting from an enumeration of all components in a system, the usage, saturation, an error statistic of each component are investigated for notable events. These events are correlated with the performance issues of the complete system. The USE approach is in stark contrast to other widely used approaches in performance analysis, which are often top-down, black-box, and ad-hoc methods. This approach resonates with the findings of the literature research on server as well as network virtualization: similar pragmatic approaches seem to be the state-of-the-art in academia and industry. In contrast to that, we argue that the SDN data plane should not be treated as a black box. We investigate the data plane as a glass box and investigate which features the SDN protocol must support to enable efficient and controllable virtualization of the individual components of data plane elements.

While the performance analysis as conducted by the USE method is related to the virtualization approach d proposed in this chapter, their goals differ. The USE method is used reactively after a performance related event occurred. Our approach is used proactively to control contention-originated performance events before they occur. Therefore, the USE approach is adapted to the needs of data plane device virtualization.

To answer Research Question 1.1 *How to characterize the control path performance in SDN data planes?*, we use an analytical bottom-up approach. The investigation starts with the actual hardware components on the control path of data plane elements that perform the operations requested through the SDN protocol by the controller. We refer to these hardware components as resources. For example, the memory interface of a flow table in a packet processing ASIC is a resource that is consumed by applications through OpenFlow messages that install flow entries.

We map these resources to the software components operating on them or interacting with them to establish causality for the usage of the resources. The software components are mapped to the OpenFlow protocol messages to establish a connection between the consumption of a resource and the messages sent by the controller. All resources affected by an OpenFlow protocol message are determined by using the controller as the starting point and the destination resource as the target and iterating over the resources that the message passes on its way. This path is termed the resource path. For example, an OpenFlow message that installs a flow table entry is processed by the management

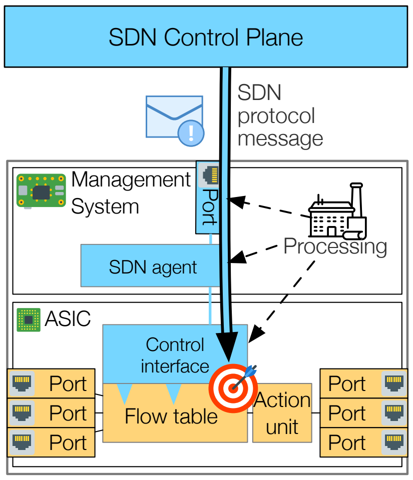


Figure 4.4: The processing of an SDN protocol message along the control path of a data plane element.

system first, then forwarded to its actual destination resource, the hardware flow table memory interface as depicted in Figure 4.4.

The controller uses the resource topology to determine the resources affected by each OpenFlow message. The resource topology comprises all software components and their connections to the resources. The resource topology is provided by the data plane elements to the OpenFlow controller, for example through a mechanism like OF–CONFIG [ONF14a]. The data plane elements provide frequent updates on the utilization level of their resources. Thereby, the controller knows all potential resource bottlenecks and their status. Furthermore, by using the resource graph, it can determine whether a message will be affected by a specific resource being overloaded or not. Using this information, the controller implements matching virtualizers for each of the resources that are directly addressed by OpenFlow messages. Resources that are not directly addressed are monitored, and their overload is prevented by controlling the load of all SDN messages that consume them.

One positive side-effect of this approach is that it makes the resource consumption of control plane applications measurable, which makes them in turn comparable for their efficiency.

4.2.2 *Data Plane Resources*

In the specific context of performance analysis, we need an abstraction that picks only the parts of the hardware that causes performance variability, and that is influenced by the workload. Hardware units, when executing a program, are limited in their output. This means that every program instruction performed on a hardware unit „consumes“ a part of the potential output of the hardware unit. For example, control plane applications can only influence the packet processing behavior when packet matching memory is available on the corresponding data plane device. If one control plane application uses all available memory slots, they are not available to other control plane applications anymore. These applications will not be able to influence the traffic passing through this specific device.

Control plane applications use these hardware units of the data plane's control path by communicating through an SDN protocol. The interaction of control plane applications with the data plane is relayed through an SDN controller, which can either grant exclusively or time-shared access to the hardware units on the data plane devices. Therefore, when executing a control plane application, these hardware units are the limiting factor for the performance of control plane applications. We investigate the hardware units and their output from a consumption-oriented perspective and therefore term them resources.

We are not the first to use the term „resource". The IETF uses this term in the context of resource locators on the Internet, but does not give a precise definition for it: „A resource can be many things." [RFC1736]. The ONF uses the term in the OpenFlow Switch Specification document [ONF15] as well as in architectural documents, e.g. [ONF14c]. However, as with the IETF, the term is not well defined: „Anything that can be used to deliver a service." [ONF14c]. The scientific literature proposed, to the best knowledge of the author, no clear definition of a resource in the context of computing as well.

In the context of this resource-oriented view on networking hardware, we approach resources as performance-oriented abstractions of the data plane hardware. This abstraction, with the term abstraction used in the sense of emphasizing relevant features of an object, aims to support the reasoning about the performance of the hardware. Therefore, the only parts of the hardware that are relevant are those that perform functions that are related to the control path of data plane devices. Performance is understood as the result of the execution of fixed or programmable logic with the help of resources, which are consumed in the process.

Definition 1: Resource
A resource is a functional part of a hardware device that is of limited abundance, consumed through or by the execution of a program on the hardware device, and influences the output of the device.

For example, a packet matching memory table that classifies packets can perform a given number of lookups per time interval. When the maximum number of lookups in an interval are conducted, additional lookups cannot be performed. Therefore, the packet matching memory table is classified as a resource, because it is limited, it is consumed by the workload, i.e., by processing packets, and it influences the processing of other packets. In contrast, the thermal throttling of a processor due to insufficient cooling is not classified as a resource. While thermal throttling influences that performance, the throttling is not caused by its consumption through a program, and therefore is not a resource.

Table 4.1: Resource characteristics.

Characteristic	**Example values**	**Description**
Resource name	CPU	–
Function	Executing of instructions	Description of the purpose of the resource.
Metric	CPU time, instructions executed	The perspective how the resource is abstracted.
Type	non-renewable, renewable	Non-renewable resources are explicitly allocated and released; renewable resources replenish over time, e.g. I/O operations.
Unit	ms, entries, kb	Unit of measure.
Abundance type	static, dynamic	The abundance of resource can be static, i.e. it has a fixed value, or variable i.e. the abundance of the resource varies over time.
Abundance	500ms, 100 entries	The abundance of the resource in units.
Allocation granularity	no, 1ms, 1 entry	The minimum abundance measured in units that can be allocated if this is possible.
Location	–	Location in the resource topology: specifies the dependencies of the resource.
Saturation	no, measured in units, measured in percent of abundance	The resource provides a work queueing mechanism of which the queue length can be measured.
Errors	no, yes, error types	The resource provides a method to collect information related to errors that occur during the operations.

Resources can have different characteristics, which are listed in Table 4.1. Packet match memory is assigned to a single control plane application and cannot be used otherwise until it is released–which is why this type of resource is called non-renewable. I/O operations, on the other hand, are renewable resources. A packet matching memory controller has a limited amount of operations it can conduct per unit of time. This means that after one application has completed its operation, the next can take over, without the first one having to release the resource. Furthermore, the granularity of resources might be different. Packet match memory might only be assigned in chunks of a certain size, e.g., 100kb, while the statistical utilization of matching memory updates might be a fraction, depending on the measurement detail.

4.2.3 *Resource Discovery and Analysis*

So far, no systematic approach exists to identify and enumerate all performance-relevant resources in heterogeneous SDN data planes. To that end, we present an approach to discover all resources and then determine if they are performance-relevant. The first step is to create a resource list from a functional block diagram of the hardware of a data plane device. Data plane devices usually consist of a management system CPUs and a packet processing ASICs. The control management system provides access to manage and control the device. For this purpose, it provides at least one NICs to connect to the control network that provides connectivity to the SDN controller. The purpose of the ASICs is to process packets passing on the data path. They are controlled and configured by the management system. In SDN the control CPUs are controlled remotely through an SDN protocol.

Therefore, after identifying all resources on the hardware level, their usage must be mapped to the corresponding SDN primitives. However, the SDN, or in this case OpenFlow software itself operates on the management system and accesses the ASIC through the management system. Therefore, we identified three layers that need to be mapped:

- OpenFlow messages
- Management system software: OpenFlow agent, operating system, and a driver for the ASIC
- Hardware: ASIC and management system

The fact that each of the layers uses different models to access the hardware resources makes the process challenging. A depiction of these three layers in the context of all modeling layers in SDN is shown in Figure 4.5.

The result of this process is a directed acyclic graph that reflects which OpenFlow operations consume which resources. We call this tree the resource topology.

Definition 2: Resource topology
The graph of resources that specifies how programs/messages/instructions reach the resource they are designed to affect, starting from its source.

The resource topology is the basis for the resource path, which is used to determine the resources affected by a given OpenFlow message. An OpenFlow protocol message that instructs the data plane element to modify a packet match memory entry must pass the network link between the control and the data plane before arriving at the management system of the data plane device. There, it is processed and forwarded to the hardware flow table, where it takes its effect.

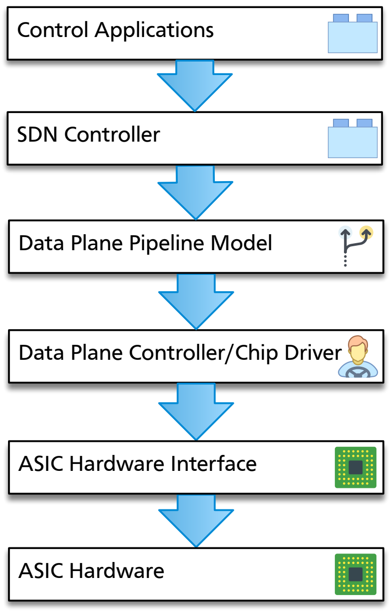


Figure 4.5: The modeling hierarchy of OpenFlow-based SDN.

Definition 3: Resource path
A resource path is the path through a resource topology a program/message/instruction moves along from the SDN controller to affect its destination resource.

The resource path is an essential concept for determining the approach to virtualize a resource and is discussed in Section 4.2.4.

Some characteristics of some resources are available to the OpenFlow controller through the OpenFlow protocol. In this case, we will still include the resource in our graph, but will not investigate them in detail.

Not all resources are relevant. From the identified resources, the ones that are not directly or indirectly affected by the OpenFlow protocol can be removed from the resource topology. For every OpenFlow message, a list of the affected resources must be investigated for bottleneck redundancy. Only resources that are potential bottlenecks need to be investigated further. Potential interactions between the different resources on the respective path must be described.

Along its path from the controller to the destination component in the data plane element hardware, the processing of the OpenFlow message will consume resources. Each of these resources could be a bottleneck. The goal of the resource topology is to enable the controller to determine which of the resource is the most likely bottleneck. The bottleneck could the destination resource, but also any other resource on the path.

4.2.4 *Resource Virtualization*

The goal of this process is to control interference on the control path by virtualizing all relevant control path resources. As proposed by the ONF [ONF14c], and having identified all relevant data path resources, we apply resource virtualization to all of them. Before discussing the details on how we achieve that, the term virtualization is defined in this context. The ONF defines the term virtualization: „The abstraction of particular underlying resources, whose selection criterion is the allocation of those resources to a particular client, application, or service.“ [ONF14c] This definition is too generic to be used in our discussion. Therefore, we define resource virtualization as follows:

Definition 4: Resource virtualization
Virtualization provides controlled, shared access to a single resource from multiple consumers of the resource that have no knowledge of each other and do not necessarily behave cooperatively.

To virtualize control path resources in the data plane, the controller needs to know the current status of the resource and has to be able to control its future workload. We term the knowledge of a resource's status resource visibility and the ability to control its workload resource controllability.

Definition 5: Resource controllability
A resource is called controllable by the SDN controller if its operational status can be influenced either by directly controlling the resource or its workload.

We understand resource visibility by the controller in terms of being aware of all resources described in the form presented in Table 4.1 and receiving timely status information.

Definition 6: Resource visibility
A resource is called visible by the SDN controller if its operational status information such as its utilization, saturation, and errors are available.

Following Gregg's work on performance bottlenecks [Gre13], we use its terminology to describe the information needed to capture the status of a resource:

Definition 7: Resource utilization
The utilization of a resource is the ratio of its used abundance to its available abundance [Gre13].

Definition 8: Resource saturation
A resource is called saturated if its workload is higher than its processing capacity [Gre13]. Saturation can either be a binary state when new work is immediately dropped by the resource or a level when new work is queued for later processing.

Definition 9: Resource errors
Resource errors are the number of error events that happened during the resource's operation [Gre13].

The relation of these terms is depicted in Figure 4.6. Resource visibility and resource controllability on all relevant resources in the data plane must be provided to the OpenFlow controller in order to be able to provide virtualization to control plane applications. To that end, the resource topology including the resource characteristics on all data path resources is provided to the controller by the data plane elements. We propose adding this functionality to the OF-CONFIG protocol [ONF14a]. However, we consider its implementation future work, since the specific transmission method is not essential for our approach. In addition to the one-time transmission of the resource topology, a facility is required that regularly provides information on each resource's utilization, saturation, and errors to the controller. When the saturation of a resource crosses a configurable threshold, the data plane element should increase the reporting rate for this resource to support the virtualization process on the controller. We suggest implementing this as an extension to the OpenFlow protocol, but again do not consider the specific implementation of this feature crucial to this discussion.

To enable the controller to provide fine-grained virtualization, we propose to implement the software on data plane elements as isolated modules that are responsible for different primitives of the SDN protocol. Each of these modules is responsible for an area of SDN primitives, such as statistics collection, flow table management, or group table management. Furthermore, like the approach proposed by Sköldström [SY12], each module is isolated, e.g., through standard OS isolation facilities to use a restricted set of

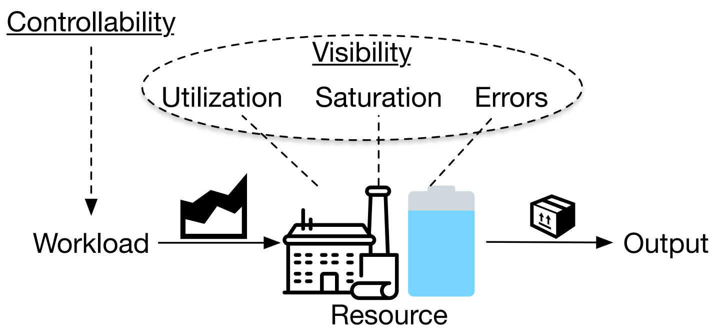


Figure 4.6: Visibility and controllability of resources.

resources, e.g., CPU cores or memory. Sköldström proposes to create isolated modules per network tenant or application, which has scalability issues. However, by creating isolated modules per SDN primitive, the number of isolated modules is fixed, and the SDN controller can provide the virtualization as required. This process enables the grouping of software and resources in the resource topology.

The approach to providing fine-grained resource isolation for the software running on the management system is inspired by Ousterhout et al., who propose the concept of „Performance clarity as a first-class design principle“ [Ous+17]. Not only increases this separation the level of control of the controller over the individual data plane devices. It also enables the collection of resource usage statistics per OpenFlow primitive. Statistical data helps in cases when the abundance type is dynamic, or the abundance is unknown. The same is true when the units of different resources cannot be translated into each other, e.g. when an OpenFlow message is forwarded through the management system CPU over the PCIe bus to the ASIC. The number of processed OpenFlow messages could be the unit of measure for the CPU. For the PCIe bus, the unit of measure could be bytes/s. Since we might not know the size of the corresponding message is on the PCIe bus, we must infer this value. One of the methods proposed in the literature, e.g., Tango [Laz+14] or OFLOPS [Rot+12] should be used to infer the missing information, detailed resource consumption information helps in this process.

With all resources being visible and controllable, the question arises how to share access to the resources between potentially uncooperative or faulty control plane applications while ensuring the controllability requirements are met.

In addition to providing controllability, we need the resource virtualization approach for each resource to match the requirements of the controller and the control plane applications. How and if these requirements can be met depends on the resource's characteristics. For example, it is impossible to design a virtualizer for resources with unpredictable abundance if the control plane application requires predictable behavior of the virtual resource. Furthermore, the characteristics of the virtualized resource can be adapted as needed. Some control plane applications might require OpenFlow flow table entries to be statically allocated, while others might be able to work with the fact that their flow table entries might be evicted. In the context of ISP networks, prioritization between applications is an important feature to ensure the reliable operation of the network.

Non-renewable resources can be virtualized on the controller, as shown in the literature on, e.g., the case of the number of flow table entries [Jin+15]. On the contrary, renewable resources require a queueing and scheduling mechanism to enable high throughput and virtualization at the same time. Ideally, the controller should operate the scheduler, while the queueing takes place on the data plane element. We will show how this can be achieved using existing OpenFlow primitives in Section 4.4.

The last issue to be discussed is that each OpenFlow message will consume multiple resources on the data path. A flow table update will consume resources of the management system as well as on the ASIC. In this case, the controller uses the available resource characteristics as well as the resource status information received from the data plane element to determine which resource is going to be the bottleneck. It will then focus on this resource and ignore the other affected resources. If multiple resources are a potential bottleneck, a different strategy is required. In this case, the data plane element should support prioritization tags for OpenFlow messages. Only one of the bottleneck resources can be directly addressed by the queueing mechanism that is operated on the controller. It is not possible to do so for the other bottlenecks efficiently. This is because other controller-scheduled bottlenecks would require the controller to wait for multiple feedback loops on the resource state in the data plane. This would slow down the processing on the data plane element because, for every processing step, it must wait for a command from the controller. Therefore, a priority tag should be used for the other bottleneck resources by the data plane element. A priority queueing system should implement the prioritization for every renewable resource on data plane elements. Thereby, the controller schedules the most likely bottleneck and leaves the others to the scheduling on the data plane element. The scheduling on the data plane element introduces the risk of accidental overloading a resource because of, e.g., unexpected processing times on the management system, but the likelihood of this happening is expected to be much smaller than the main bottleneck resource, this risk is considered small.

Finally, the resource virtualization approach should support the collection of statistics on the resource usage of control plane applications. To achieve resource efficiency, the resource consumption of control plane applications has to be measurable.

4.3 A CONTROL PATH RESOURCE MODEL OF AN OPENFLOW DATA PLANE ELEMENT

In this section, the resource identification approach described in Section 4.2 is applied to a representative OpenFlow switch architecture. The Edge-Core AS5712-54X is selected and investigated. It is a state-of-the-art bare-metal 10GbE switch based on the Broadcom Trident II switching ASIC. It is operated by the PicOS operating system version 2.8 that uses Open vSwitch version 2.3 [Pfa+15] as its OpenFlow agent.

An overview of OpenFlow switches used in literature as well as why the Edge-Core AS5712-54X was chosen for this investigation is discussed in Section 4.3.1. The soft- and hardware of the management system of the Edge-Core AS5712-54X are analyzed in Section 4.3.2. A resource model for the Trident II is derived in Section 4.3.3. The resulting, complete resource list for the Edge-Core AS5712-54X including the corresponding resource topology is presented in Section 4.3.4.

Table 4.2: Overview of investigated OpenFlow switches in academia.

Model	**ASIC/NPU**	**CPU**	**OS**	**Cita-tions**	**Comment**
Arista 7050	Broadcom Trident+	AMD Turion II Neo N41H	Arista EOS (Linux)	1	End of life
EdgeCore AS5712-54X	Broadcom Trident 2	Intel Atom C2538	PicOS (Linux)	1	-
NEC PF5240	Broadcom unknown	PowerPC	propr. (NetBSD)	3	Closed OS
NoviFlow NoviSwitch 1132	EzChip NP4	Intel Core i7-620LE	NoviWare (Linux)	1	Not sold anymore
Pica8 P-3290	Broadcom Firebolt 3	NXP/Freescale MPC8541E	PicOS (Linux)	6	Not sold anymore

4.3.1 *Overview of Available OpenFlow Data Plane Elements*

We surveyed available OpenFlow switches in the industry and their usage in academia in Section 3.1.4. We discuss selected devices as listed in Table 4.2 from this survey for their representativeness for state-of-the-art SDN data plane elements and reason which we selected Edge-Core AS5712-54X switch for this investigation. None of the identified SDN devices is designed to be used in ISP core networks. However, we expect that adequate devices for ISP core networks will be available and that our methodology can be applied to these devices as well.

The NoviFlow NoviSwitch 1132 is an interesting device. However, it is not sold anymore and based on an NPU, and thereby not representative for the whole class of switches, which are mostly driven by ASICs. Interestingly, the remaining devices in our selection, as well as most of the devices investigated in academia rely on Broadcom ASICs. Devices from HP are popular in literature but are not considered here, because both their hardware and software are not state-of-the-art. The most often used devices in academia are sold by Pica8, represented by the P-3290 in the table, followed by NEC represented by the PF5240. The devices from NEC use a closed source, proprietary OS that makes it difficult to investigate the software architecture in detail. The PicOS OS used by the P-3290 is interesting, because it is Linux-based, and open to the operator, except for the proprietary ASIC driver. However, the hardware of the P-3290 is outdated, and the device is not sold anymore. Arista uses Linux as a basis for its EOS operating system, but unfortunately, the 7050 series that was used in academia before, was announced by Arista to be end-of-life, which makes the device obsolete.

The remaining device that was investigated in academia before is the Edge-Core AS5712-54X, which is based on a recent Broadcom Trident II ASIC design and runs the Linux-based PicOS. The device is still sold, its management system relies on an up-to-date Intel CPU, and PicOS is available for it and is actively supported. While the driver for Broadcom devices that comes with PicOS is proprietary and not available as open source, the operating system is. The only alternative OpenFlow agent is the OpenFlow - Data Plane Abstraction (OF-DPA), which also provides a proprietary, closed source driver for a range of popular Broadcom ASIC series and documentation for this model. It is available as a binary for open-source operating systems. However, while the OF-DPA software relies on the OpenFlow protocol, it does not adhere to the OpenFlow pipeline model. Therefore, OF-DPA cannot be easily used by existing OpenFlow software, and it cannot be considered state-of-the-art. The Edge-Core AS5712-54X with PicOS includes a state-of-the-art packet processing ASIC, management system, and OpenFlow agent software. Furthermore, its OS is Linux-based and completely open to the operator. Therefore, we use this device for our in-depth investigation and the representative creation of a resource topology.

4.3.2 *Investigating the Soft- and Hardware of the Edge-Core AS5712-54X Management System*

The Edge-Core AS5712-54X is a state-of-the-art bare-metal 10GbE switch based on the Trident II switching ASIC, specifically the Broadcom BCM56854. Bare-metal switches are defined by the fact that they are sold without a pre-installed operating system. In this case, the switch operated by the PicOS operating system version 2.8 that uses Open vSwitch version 2.3 [Pfa+15] as its OpenFlow agent frontend and a proprietary backend to interface with the ASIC. PicOS supports different OpenFlow modes, of which the default mode is investigated. It supports the standard OpenFlow model with a single flow table.

In addition to the Edge-Core AS5712-54X, we will complement the investigation with four other switches to compare, contrast, and assess the Edge-Core AS5712-54X's architecture. The devices used for the investigation are listed in Table 4.3 in descending order of relevance to this investigation. The Arista 7050 is, like the Edge-Core AS5712-54X, a 10GbE switch. However, it is an older model and is more restricted when it comes to installing third party software on the management system. Still, it is useful to compare it to the Edge-Core AS5712-54X. The Delta Networks device is based on the Barefoot Tofino ASIC, a new generation of programmable data devices that can be programmed using the P4 programming language. Unfortunately, it is currently only available as a developer switch that comes with a software development environment and a corresponding non-disclosure agreement (NDA) that prohibits the disclosure of detailed results. Still, it represents the latest development in data plane devices, which is helpful to understand the direction of development in the industry. Its management system design with an eight-core server processor, e.g., shows that the performance of the management system-CPU can be increased in future device designs, if necessary. The NEC PF5240 represents the first generation of switches with traditional switching ASICs but with software that is purpose-built for SDN. Finally, the HP 3500 represents one of the first generations of switches that were retrospectively upgraded with SDN functionality.

Table 4.3: Overview over OpenFlow switches available for this thesis.

Model	ASIC	ASIC-CPU Interface	CPU	Cores	RAM	OS	Switch softw.
EdgeCore AS5712-54X	Broadcom Trident 2	PCIe 2.0, 2 lanes	Intel Atom C2538	4	8GB	PicOS (Linux)	PicOS
Delta Networks ET-X064FFRB	Barefoot Tofino	NDA	Intel Xeon D 1548	8	32GB	Ubuntu (Linux)	Bare-foot SDE
Arista 7050	Broadcom Trident+	PCIe 2.0, 2 lanes	AMD Turion II Neo N41H	2	4GB	Arista EOS (Linux)	Arista EOS
NEC PF5240	Broadcom unknown	unknown	PowerPC	un-known	1GB	propr. (NetBSD)	propr.
HP 3500 yl	HP ProVision	PCI/ PCI-X	NXP/ Freescale MPC8540	1	256MB	propr.	propr.

However, it uses an outdated management system with a very slow CPU and uses the by today's standard slow Peripheral Component Interconnect (PCI) bus for connecting the ASIC. Results gathered on similar HP devices in literature must be carefully analyzed to separate the impact of the outdated management system from findings that can be generalized to other SDN devices.

This analysis of the Edge-Core AS5712-54X starts with investigating the software that interacts with the OpenFlow controller and is running on the management system of the switch followed by an investigation of the hardware resources of the management system itself. A detailed view of the architecture of the Edge-Core AS5712-54X's management system running PicOS is given in Figure 4.7. In the depiction the separation between the Linux OS, the OpenFlow agent component of Open vSwitch, ovs-vswitchd, and the interface to the ASIC are visible. In addition to these software components, the hardware components the software operates on are depicted. PicOS relies on Linux and Open vSwitch to accept incoming OpenFlow messages on the management NIC that is part of the management system. The messages arrive in the Linux network stack and are forwarded to ovs-vswitchd. There, the message is translated from OpenFlow to an internal format, and the result of the message is stored in a data store. Then, the change in the data store is identified by another process that translates it to a PCIe message that is forwarded to the ASIC.

Most of this process, except for the last step, is performed by the management system of the switch. The software components involved in the process are the Linux kernel, the ovs-vswitchd component of Open vSwitch, and the ASIC interface. The Linux kernel

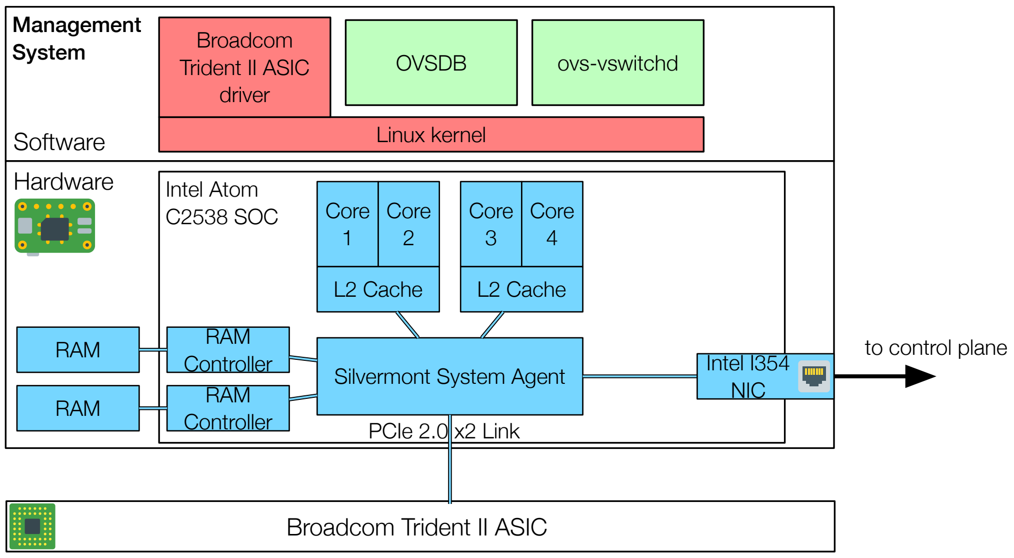


Figure 4.7: The soft- and hardware architecture of the Edge-Core AS5712-54X running PicOS 2.8 (based on Intel Atom and PicOS documentation[3] and [Pfa+15]).

is responsible for supporting the other software components and is involved when the other components interact with the management NIC and the ASIC as part of the processing of incoming and outgoing OpenFlow messages. Ovs-vswitchd accepts and parses OpenFlow messages and translates them into messages that are sent to the data path ASIC. The Open vSwitch software package is has been adapted by Pica8 to interact with the Trident II ASIC as well as with a software switch running on the same CPU. However, these modifications are not available as open source software, which is why in-depth documentation on their functionality and architecture is not available.

What is possible though, is to infer the possible approaches from the functionality of the software and its publicly available documentation[4]. To understand the interaction between the Open vSwitch software and the Trident II ASIC the corresponding commands available in the software are investigated. It shows that by default, the Pica8 version of Open vSwitch will accept any OpenFlow message the Open vSwitch accepts, independent of the fact whether the Trident II ASIC supports the packet handling features contained in the message. It the packet handling is not supported by the ASIC the flow entry will be installed in the Open vSwitch software data path operating on the management system. Corresponding packets then seem to be forwarded from ASIC to the management system where they are passed through the software data path, and the result is sent

3 Pica8. *PicOS Open vSwitch Configuration Guide*. Version 1. Jan. 2017.
David Mulnix. *Intel Atom Processor C2000 Product Family Technical Overview*. Accessed: 2018-9-12. Intel, Sept. 2013. url: `https://software.intel.com/en-us/articles/intel-atom-c2000-processor-family-technical-overview`.

4 Pica8. *PicOS Open vSwitch Command Reference, PicOS 2.8*. Version 1. Jan. 2017.

back to the ASIC for forwarding the packet to the output port if needed. The software data path is either the user space data path provided by Open vSwitch or proprietary software. This functionality is disabled in our configuration because it is not useful in a high-performance ISP network, as discussed in Section 4.1.2. The fact that this software forwarding process works seamlessly indicates that OpenFlow messages are always applied to the software data path first and to the hardware data path, i.e., the ASIC only where applicable. Still, the flow tables of the software and hardware data path can be displayed separately by the Pica8 Open vSwitch command line interface (CLI).

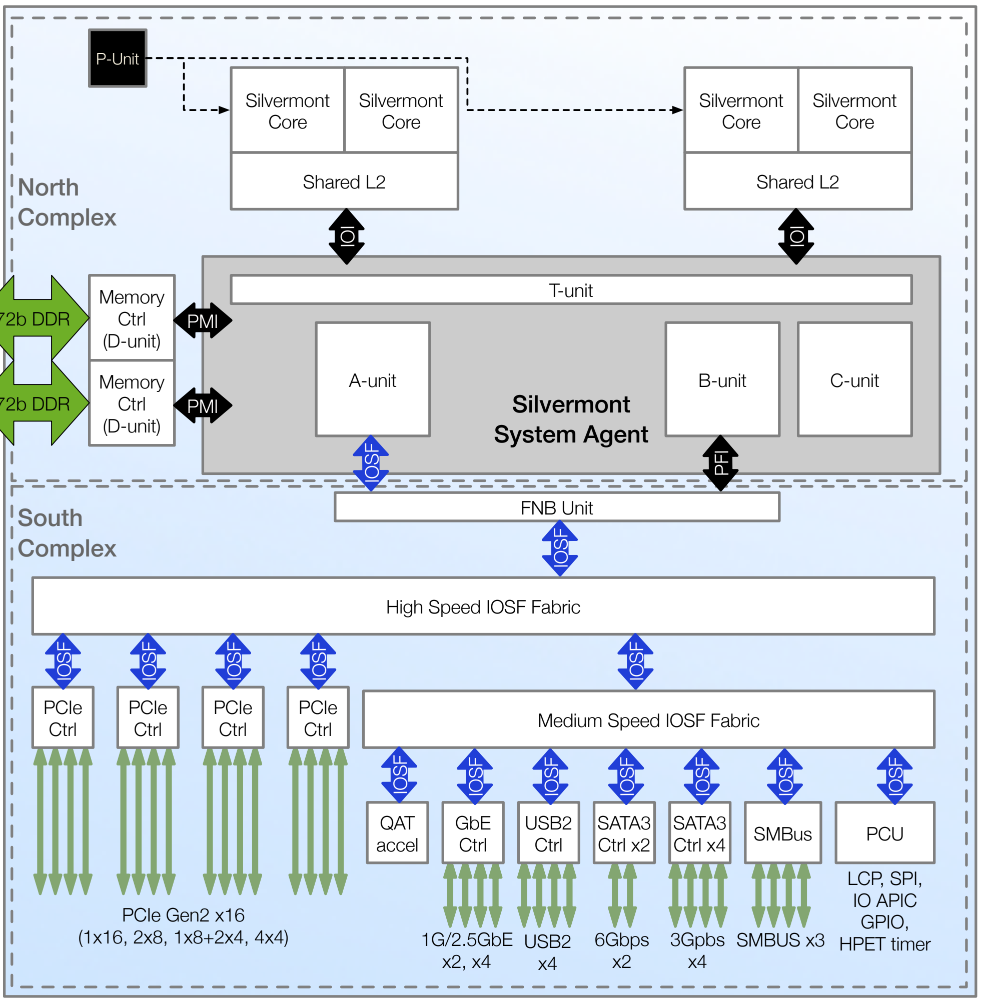


Figure 4.8: Block diagram of the Intel Atom C200 family microarchitecture (adapted from the Intel Atom Processor C2000 documentation[5]).

5 David Mulnix. *Intel Atom Processor C2000 Product Family Technical Overview*. Accessed: 2018-9-12. Intel, Sept. 2013. url: https://software.intel.com/en-us/articles/intel-atom-c2000-processor-family-technical-overview.

In literature, e.g., Kuzniar et al. [Kuz+18] it has been reported that barrier_reply messages sent from an OpenFlow switch running the PicOS software do not indicate that all previous OpenFlow messages are completely processed in the data path as is specified in the OpenFlow specification. In contrast, these messages are sent by the switch even though some of the previous flow table modifications of the devices' flow table are not active in the ASIC yet. This observation, together with the usage of the software data path could mean that the OpenFlow processing is oblivious of the ASIC data path and instead provides feedback for the software data path only.

Sending packets to and receiving packets from the ASIC requires processing by the ASIC driver that then forwards the packet to the ovs-vswitchd component. There, the packet is either processed locally or forwarded to the OpenFlow controller through the management NIC.

We conclude the discussion of the software operating on the management system by summarizing the significant tasks of the software running there:

- creating and maintaining a network communication connection with the OpenFlow controller
- OpenFlow message processing for messages received from and sent to the OpenFlow controller
- translating OpenFlow messages to ASIC messages and vice versa
- PCIe communication with the data plane ASIC

The Edge-Core AS5712-54X uses an Intel Atom C2538 system on chip (SoC). A SoC integrates multiple chips into one, in this case, the chip includes CPUs, memory controllers, north- and southbridge, as well as a NIC. To the best knowledge of the author, no resource analysis has been published for this CPU architecture. Based on a resource analysis that has been conducted by the author for an Intel Xeon E5-2600 v3-based system [Ble+16a], we analyze the resources of the Intel Atom C2538 CPU. The block diagram of the Intel Atom C200 family CPU microarchitecture is depicted in Figure 4.8. Each CPU core is a resource. The L1 cache is dedicated to each CPU core but can be shared between programs running on the same CPU. Each pair of CPU cores shares the 1MB L2 cache. They are consumed by processes running on them and the amount of available cache, no matter at which level can significantly affect the performance of a program. Research shows that for example, uncontrolled shared caches in CPUs cores can reduce the performance of neighboring processes [Koh+07]. The same goes for the interconnection facilities, namely the Silvermont System Agent and the connected fabric and PCIe busses. Finally, the memory controllers can process a limited amount of transactions per time–again they must be regarded as resources. The result of this analysis is the detailed resource model that is depicted in Figure 4.7.

However, there is no indication that resource limiting, or controlled sharing is used by PicOS to separate the processes running on the management system. Therefore, a direct mapping between the software processes and the hardware resources is not possible. The Linux kernel and the Debian Linux OS distribution PicOS is based on include the necessary kernel and user space tools to do so, using the Linux Control Groups feature[6] that was already proposed by Sköldström [SY12]. To create a fine-granular resource model, and therefore, improve the system's performance clarity, we argue that a resource separation would be advantageous.

In conclusion, the resource topology for the management system must be adjusted to match the granularity of resource accounting and sharing. Since the OS and the OpenFlow agent are not separated, the whole management system must be seen as a single resource pool as depicted in the corresponding resource topology in Figure 4.9. Links that can be used for the resource path are depicted as solid lines, resource dependencies are depicted as dashed lined. Resource dependency nodes are depicted as ovals with dashed lines; the actual resources are depicted as ovals with solid lines.

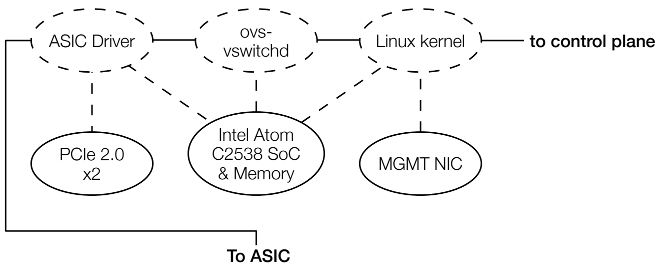


Figure 4.9: The resource topology of the management system.

4.3.3 *Investigating the Trident II ASIC*

Investigating the ASIC requires a different approach than investigating the management system. In contrast to the Intel Atom SoC, there is no documentation or architectural information available for the Trident II or any of the other ASICs of the line of ASICs that is supported by PicOS. Therefore, we approach the analysis through the documentation of the OpenFlow pipeline provided by PicOS. The pipeline model, together with the SDN protocol OpenFlow, define what kind of behavior is supported and what kind of primitives are available for control plane applications to control the packet processing.

The OpenFlow data plane model is depicted in Figure 4.10. The main functional parts inside a flow table are the matching table, the action units, and the input and output units. The actual packet processing in the data plane ASIC is modeled by OpenFlow as match-action units, referred to as flow tables. Flow tables are the central unit of abstraction in OpenFlow. The hardware of the Trident II supports multiple OpenFlow software agents that have different characteristics. The default OpenFlow mode of PicOS

6 Tejun Heo. *Control Group v2*. Accessed: 2018-9-12. Linux Kernel Organization, Oct. 2015. url: https://www.kernel.org/doc/Documentation/cgroup-v2.txt.

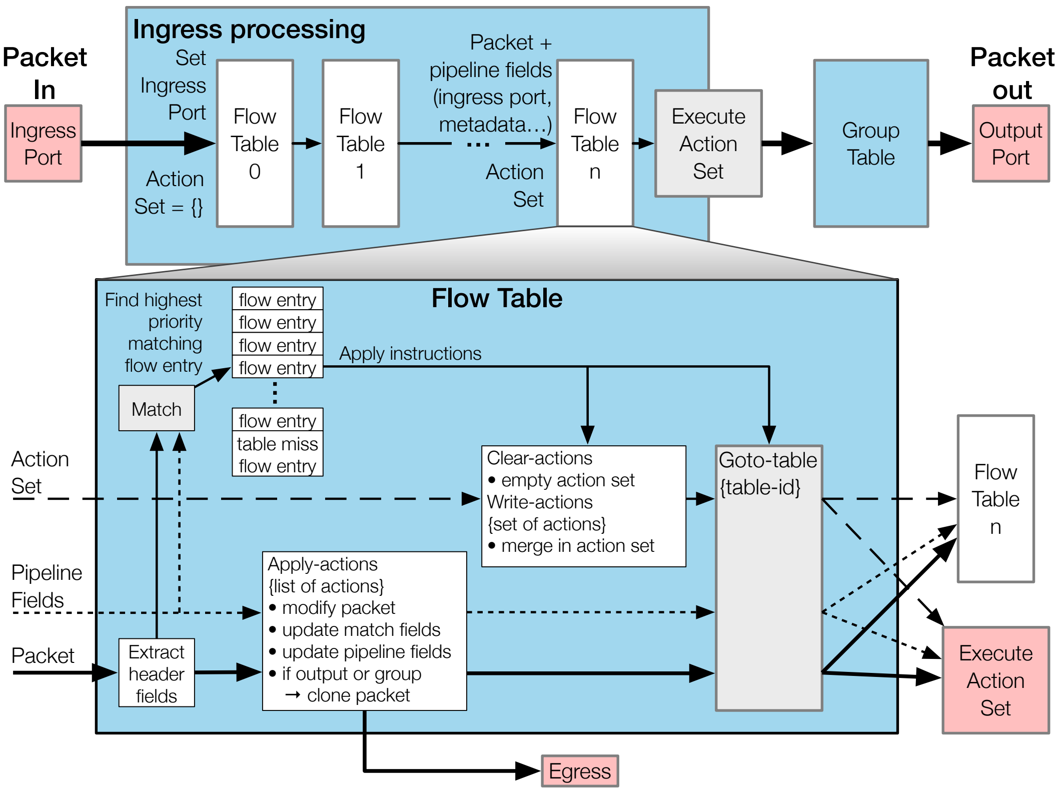


Figure 4.10: OpenFlow model of a data plane device (adapted from [ONF15]).

that relies on a single flow table, is investigated in this resource analysis. Nevertheless, analyzing different modes of OpenFlow operation is useful for discovering resources in the Trident II ASIC.

Their use in the pipeline can be restricted in more detail than specified in the OpenFlow switch specification using Table Type Pattern (TTP) [ONF14b] or OF-CONFIG [ONF14a]. These restrictions enable OpenFlow to model hardware data planes, where, e.g., some tables offer only a subset of the OpenFlow functionality, e.g., a limited set of match fields or actions. The TTP model of the Edge-Core AS5712-54X with PicOS is depicted in Figure 4.11. Table 60, the *Policy ACL* table is the only table that offers the full match and action feature set that is supported by the Trident II and is based on TCAM memory. All the other tables offer a specific and limited OpenFlow feature set.

Furthermore, the pipeline makes heavy use of group tables. Specifically, every packet is applied to at least one of the tree *indirect* type group tables *L3 Unicast*, *L2 rewrite*, or *L2 interface*. Consequently, every output action written to the main flow table results in an additional write to one of these group tables.

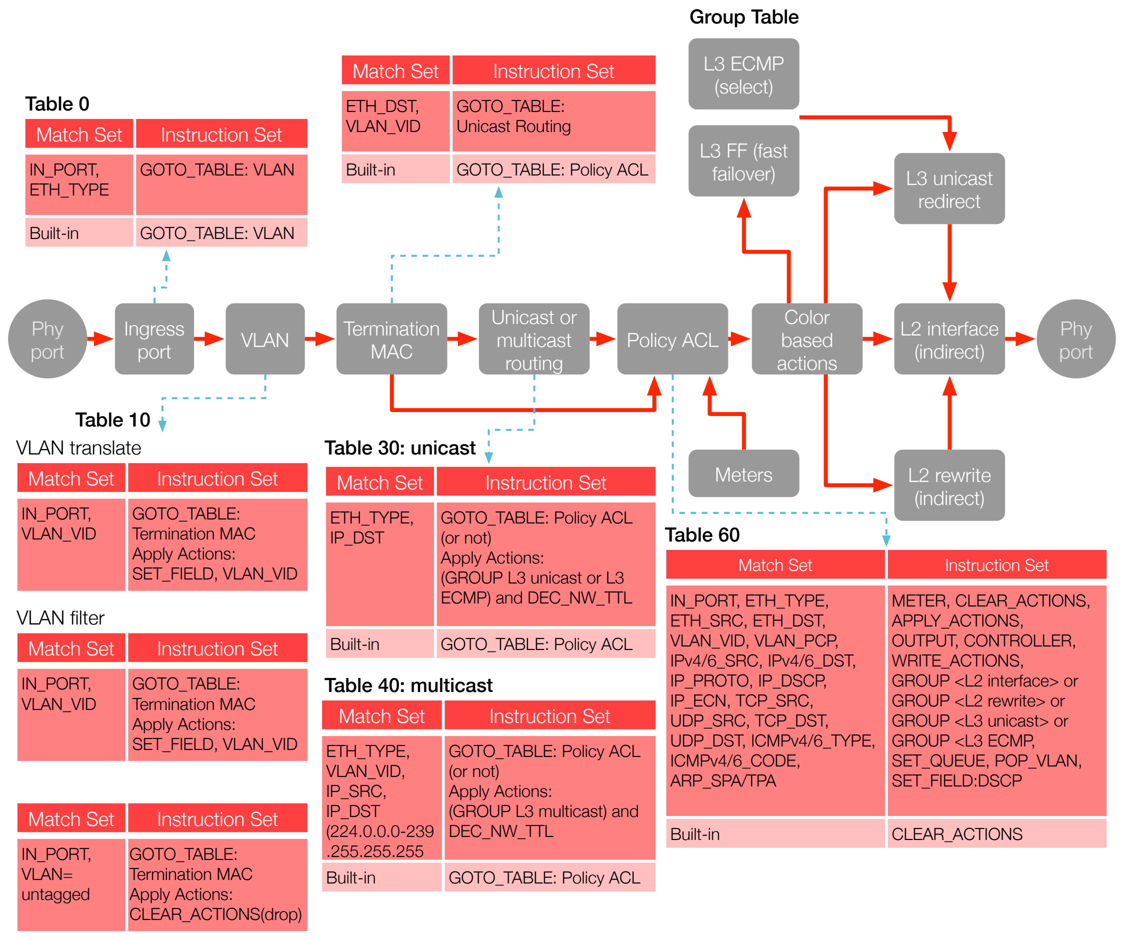


Table 0

Match Set	Instruction Set
IN_PORT, ETH_TYPE	GOTO_TABLE: VLAN
Built-in	GOTO_TABLE: VLAN

Match Set	Instruction Set
ETH_DST, VLAN_VID	GOTO_TABLE: Unicast Routing
Built-in	GOTO_TABLE: Policy ACL

Table 10

VLAN translate

Match Set	Instruction Set
IN_PORT, VLAN_VID	GOTO_TABLE: Termination MAC Apply Actions: SET_FIELD, VLAN_VID

VLAN filter

Match Set	Instruction Set
IN_PORT, VLAN_VID	GOTO_TABLE: Termination MAC Apply Actions: SET_FIELD, VLAN_VID

Match Set	Instruction Set
IN_PORT, VLAN= untagged	GOTO_TABLE: Termination MAC Apply Actions: CLEAR_ACTIONS(drop)

Table 30: unicast

Match Set	Instruction Set
ETH_TYPE, IP_DST	GOTO_TABLE: Policy ACL (or not) Apply Actions: (GROUP L3 unicast or L3 ECMP) and DEC_NW_TTL
Built-in	GOTO_TABLE: Policy ACL

Table 40: multicast

Match Set	Instruction Set
ETH_TYPE, VLAN_VID, IP_SRC, IP_DST (224.0.0.0-239 .255.255.255	GOTO_TABLE: Policy ACL (or not) Apply Actions: (GROUP L3 multicast) and DEC_NW_TTL
Built-in	GOTO_TABLE: Policy ACL

Table 60

Match Set	Instruction Set
IN_PORT, ETH_TYPE, ETH_SRC, ETH_DST, VLAN_VID, VLAN_PCP, IPv4/6_SRC, IPv4/6_DST, IP_PROTO, IP_DSCP, IP_ECN, TCP_SRC, UDP_SRC, TCP_DST, UDP_DST, ICMPv4/6_TYPE, ICMPv4/6_CODE, ARP_SPA/TPA	METER, CLEAR_ACTIONS, APPLY_ACTIONS, OUTPUT, CONTROLLER, WRITE_ACTIONS, GROUP <L2 interface> or GROUP <L2 rewrite> or GROUP <L3 unicast> or GROUP <L3 ECMP, SET_QUEUE, POP_VLAN, SET_FIELD:DSCP
Built-in	CLEAR_ACTIONS

Figure 4.11: OpenFlow model of a Trident II-based plane device running PicOS in TTP mode (adapted from the PicOS documentation[7]).

TCAM-based Flow Tables

TCAM is used to implement the single OpenFlow flow table available in the PicOS operating mode investigated here. This approach is to be expected because the original OpenFlow design modeled matching memory to have the capabilities of TCAM memory. TCAM is used because its high lookup speed and its ability to tag each entry with a priority as well as to include *don't care* bits in the match patterns, which effectively enables masked matches. One important use case for masked matches and entry prioritization is the longest prefix match used for IP routing. One disadvantage of TCAM, besides the high price and power consumption, is that inserting new entries into TCAM-based match tables can be very slow and yields hard-to-predict completion times.

7 Pica8. *PicOS Open vSwitch Configuration Guide*. Version 1. Jan. 2017.

For illustration, a simplified, yet complete overview of the circuit-level design of a TCAM is depicted in Figure 4.12. Match values are stored as zeros, ones, and wildcards

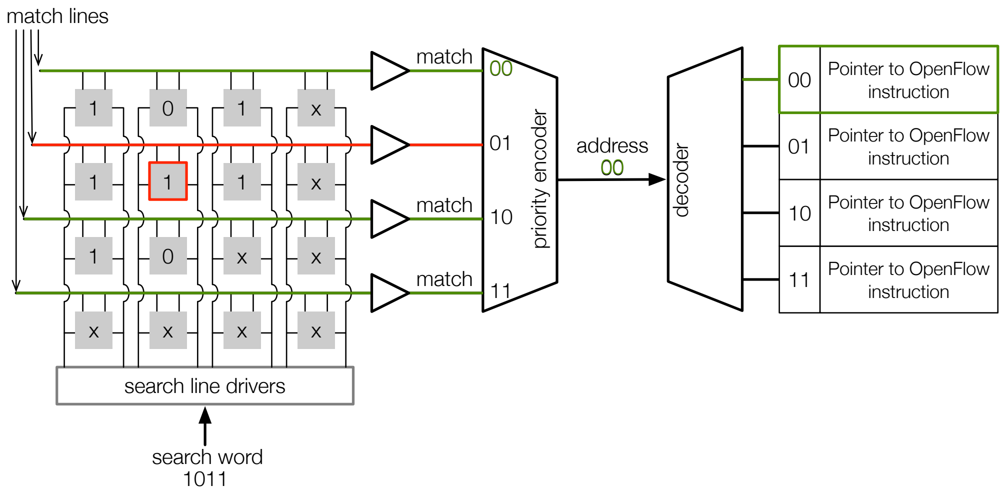

match lines
1
0
1
x
1
1
1
x
1
0
x
x
x
x
x
x
match
match
match
00
01
10
11
priority encoder
address
00
decoder
00
01
10
11
Pointer to OpenFlow instruction
Pointer to OpenFlow instruction
Pointer to OpenFlow instruction
Pointer to OpenFlow instruction
search line drivers
search word
1011

Figure 4.12: TCAM match example (adapted from [PS06]).

in the matching entries of the TCAM. The search word is compared to all table entries in parallel in one cycle, which results in every matched entry to activate an input line for the priority encoder that derives the resulting match entry. Match entries that do not match are not activated. The priority encoder selects from all activated entries the one with the highest priority. The resulting entry index is passed to an accompanying SRAM memory, where a reference to the resulting OpenFlow instruction is stored.

The reason for the slow and unpredictable insertion rate for new entries in TCAM is that the priority of table entries is encoded in the circuit by the order and location every entry is physically stored in. Therefore, when adding an entry with a higher priority than existing entries, it might be necessary to move all existing table entries to new locations. Hence, not only the priority of new entries and their order but also the priority and the number of existing entries in a TCAM-based match table influence the amount of time it takes to complete an addition operation. When the instruction of a flow entry in a TCAM-based flow table is modified, the TCAM itself does not have to be modified. Instead, the mapping of the TCAM match index to the OpenFlow instruction, which is stored in SRAM, is updated. This process does not require reordering and is therefore expected to exhibit consistent modification delay.

Measurements with TCAM-based flow tables by Chen et al. confirm our analysis [CB17b]. Flow modifications take constant time. Interestingly, so do delete operations. These observations lead to the conjecture that existing TCAM table entries can be disabled without reorganizing the whole table.

Due to the closed nature of the PicOS software, we can only speculate on how the driver software accesses TCAM memory in the ASIC. The slowness of updates and reordering suggest that the TCAM entries are accessible as an array of entries which represent a fixed matching order. Furthermore, we assume that table entries can be written individually. When a new flow entry is inserted, the driver software checks if existing entries must be moved to new locations. If this is the case, the existing entries are copied to their new positions and the new entry is written to its new location. This process would explain the unpredictable duration of table entry insertions. This also means that from the perspective of the driver, the TCAM table memory interface is a resource with a well-known and fixed amount of operations per unit of time. The unpredictable behavior that is observed from the OpenFlow controller is caused by the driver moving TCAM entries, not the write speed of the TCAM controller.

Packet Forwarding along the Control Path

Sending packets from the data plane to the control plane and vice versa are OpenFlow primitives that can provide insights into the ASIC's architecture. These primitives instruct the packet processing pipeline to forward packets to the ASIC's PCIe controller. Furthermore, it is highly likely that this virtual network port for sending packets to the switch controller is assigned a part of the shared packet buffer of the switch. The same should be the case when sending packets from the switch controller into the data plane. We measured the maximum throughput for packet_in OpenFlow messages to be 12000pps and for packet_out messages to be 7000pps. This discrepancy indicates that for packet_out neither the ASIC nor the PCIe interface is the bottleneck resources, but the management system CPU. The measured rate for packet_in could be either a CPU limitation or an actual limitation in the ASIC. Since we are not able to determine the bottleneck for these limits, we use the measured values as bounds.

Group and Meter Tables

Group tables are treated not as uniform in the Trident II ASIC as they are designed in OpenFlow. There are four types of group tables in OpenFlow: *indirect*, *all*, *fast failover* and *select*. Each group table type gets its own representation in the PicOS TTP pipeline model of the Trident II. This suggests that each group table type is implemented by different hardware units, creating a different resource each. Furthermore, two *indirect* group tables exist, one for rewriting Ethernet addresses and one for IP unicast redirect. The assumption of different hardware units per group table type is supported by the fact that only packets with a specific Ethernet or IP address can be forwarded to the group table type *all*, which is used for multicasting. While this restriction does not exist in the PicOS single table mode we analyze, we assume that this separation between the group table types was introduced to the model because of the underlying hardware. The OpenFlow agent provides the available number of group table entries through corresponding OpenFlow messages. These limitations are therefore already covered by the OpenFlow protocol. However, each of the group table units has its own memory

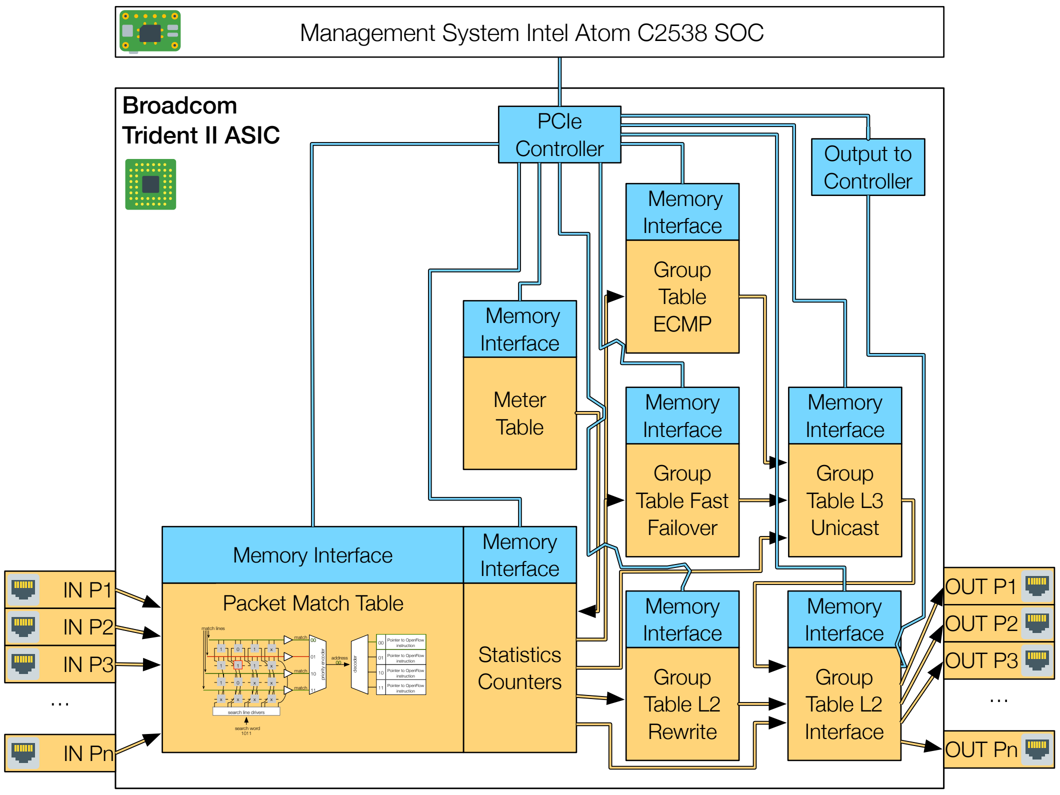


Figure 4.13: The hardware architecture of the Broadcom Trident II (based on the PicOS documentation[8]).

interface that used to read, write, and delete table entries. These interfaces are potential bottlenecks, like the TCAM memory interface described before.

ASIC Resource Overview

We conclude this OpenFlow-based analysis of the Trident II ASIC by providing a complete overview of the hardware architecture in Figure 4.13 as well as the detailed resource topology for in Figure 4.14. The lack of information on the architecture of the Trident II ASIC is reflected in the topology. The pipeline model used for the investigation is not only valid for the Trident II but a whole family of devices. This means that there are differences between the hardware implementations that are lost due to the generalization of the pipeline model. Nevertheless, the architectural model presented here is to our best knowledge the most detailed available and is depicted for completeness in Appendix Figure A.1. The corresponding resource topology is depicted in the Appendix Figure A.2.

8 Pica8. *PicOS Open vSwitch Configuration Guide*. Version 1. Jan. 2017.

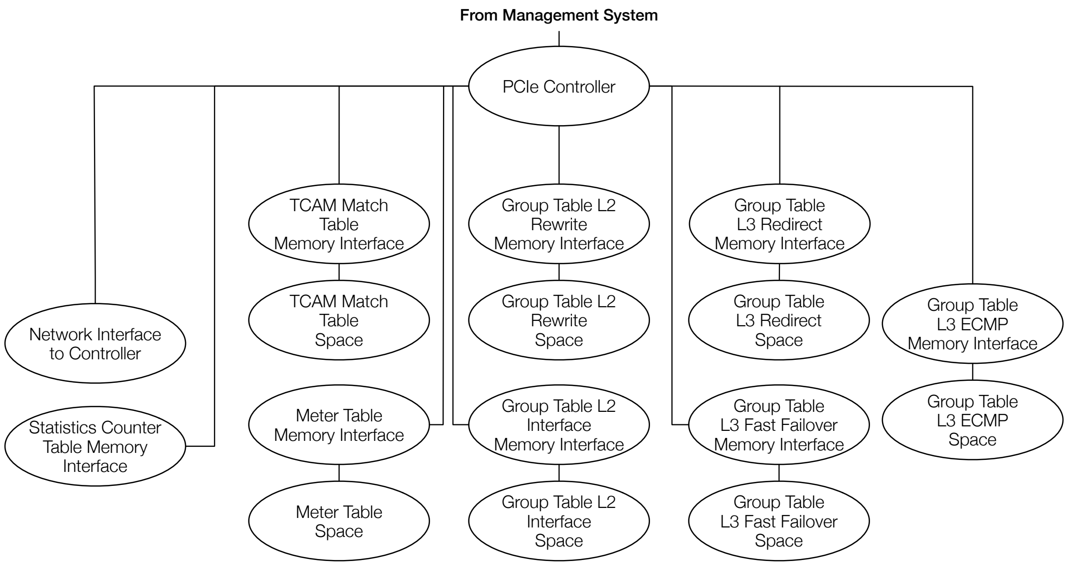

From Management System
PCIe Controller
Network Interface to Controller
Statistics Counter Table Memory Interface
TCAM Match Table Memory Interface
TCAM Match Table Space
Meter Table Memory Interface
Meter Table Space
Group Table L2 Rewrite Memory Interface
Group Table L2 Rewrite Space
Group Table L2 Interface Memory Interface
Group Table L2 Interface Space
Group Table L3 Redirect Memory Interface
Group Table L3 Redirect Space
Group Table L3 Fast Failover Memory Interface
Group Table L3 Fast Failover Space
Group Table L3 ECMP Memory Interface
Group Table L3 ECMP Space

Figure 4.14: The resource topology of the Broadcom ASIC.

4.3.4 *Mapping the Resource Topology of the Edge-Core AS5712-54X with PicOS to OpenFlow*

The next important step is to map the identified resources to the OpenFlow messages that the OpenFlow controller uses to control the data plane device. Before doing that, we adjust the resource topology to the information granularity the controller can actually see. PicOS provides no means for resource isolation between different software components and no monitoring approach by default. The OpenFlow controller has no way of discriminating which component uses which resources. Therefore, except for the NIC and the PCIe link, which are still considered separate resources, the resources are merged into a single, opaque management system SoC resource. For cases when the management system is overloaded, OpenFlow provides a mechanism to provide feedback to the OpenFlow controller: the congestion control of the control channel protocol is used to stop the controller from sending more messages [ONF15].

The resource table from the perspective the OpenFlow controller is shown in Table 4.4. For comparison, the full resource table from the perspective of the Edge-Core AS5712-54X itself is listed in Appendix Table A.1. In addition to the merging of the management system resources, the *flow table memory interface* resource has a dynamic abundance from the perspective of the controller, while it has a static abundance from the perspective of the ASIC driver.

We find that the characteristics of static, non-renewable data path resources have been well investigated in the literature. While the resource-oriented perspective provides a clear understanding of these processes, we found no resources that have been completely neglected in the past. Static aspects such as table sizes, port number and capacities are

Table 4.4: The resource table for the Edge-Core AS5712-54X with PicOS 2.8 from the perspective of the SDN controller.

Resource	**Abundance type**	**Potential Bottleneck**	**Utilization**	**Saturation**	**Errors**	**Visibility**	**Controllability**	**Allocation granularity**
Management system SoC	dynamic	yes	no	yes	no	no	yes	n/a
Management NIC	static	yes	indirect	no	no	no	yes	packet
PCI-e link	static	no	no	no	no	no	no	none
PCI-e controller	static	yes	no	no	no	no	no	none
Flow table space	static	yes	yes	units	yes	yes	yes	1 entry
Flow table memory interface	dynamic	yes	no	no	no	no	yes	none
Group table space: indirect/select/all	static	yes	yes	units	yes	yes	yes	1 entry
Group table memory interface: indirect/select/all	static	yes	no	no	no	no	yes	none
Meter table space	static	yes	yes	units	yes	yes	yes	1 entry
Meter table memory interface	static	yes	no	no	no	no	yes	none
Statistics counter table memory interface	static	yes	no	no	no	no	yes	none
Packet output to switch controller port	static	yes	no	no	no	no	yes	meter feature

already part of the OpenFlow or OF-CONFIG protocol and can be used by controllers today. The behavior of renewable resources in the control path, however, is not well understood when the resources are shared between control plane application.

We find multiple potential bottlenecks on the control path that have not been investigated for resource contention between control plane application yet:

- Packet communication along the control path: packet_in, packet_out
- Adding flow table entries to TCAM-based flow tables: flow_mod
- Modifying group tables and their group buckets: group_mod
- Modifying meter tables: meter_mod

The mapping of the selected OpenFlow messages to data plane resources are listed in Table 4.5. In the table, an x denotes a resource consumption that is likely to create a performance bottleneck, a + denotes a resource consumption that is unlikely to cause a bottleneck, and a - denotes no consumption of this resource at all.

Table 4.5: The mapping of selected OpenFlow messages to data plane resources (adapted from [Här17]).

Resource	**packet_out**	**flow_mod (insert)**	**flow_mod (modify)**	**flow_mod (delete)**
Management system SoC	x	x	x	x
Management NIC	x	x	x	x
PCI-e link	+	+	+	+
PCI-e controller	x	+	+	+
Flow table space	-	x	-	+
Flow table memory interface	-	x	+	x
Group table space: indirect/select/all	-	-	-	-
Group table memory interface: indirect/select/all	-	+	+	+
Meter table space	-	-	-	-
Meter table memory interface	-	-	-	-
Statistics counter table memory interface	-	-	-	-

The interference caused by packet_in has not received much attention in the literature but can be mitigated by applying meters before sending the packet to the control path as proposed by the OpenFlow specification [ONF15]. However, this requires the controller to know the exact capacity of packet forwarding along the control path. Instead, we propose to use priority queuing on the path to the management system as it is done on other network links. However, implementing this on a hardware switch is not possible, because of the proprietary and closed software ecosystem. We consider an implementation on programmable data planes with [P4_14] possible, but out of the scope of this work. Still, we consider the data path resource contention of packet_in a solved issue and will not investigate it in detail.

The addition of flow entries to TCAM-based flow tables has not been virtualized yet. Furthermore, the importance of this OpenFlow feature and its dynamic abundance makes it challenging to design a virtualizer. Finally, we consider the OpenFlow messages that rely on resources for which no virtualizer has been provided yet, meter_mod and group_mod, simplified cases of the flow entry addition virtualizer. Therefore, by providing a virtualizer for flow entry addition, we consider the virtualization of this class of resources solved.

4.4 VIRTUALIZING THE ADDITION OF FLOW TABLE ENTRIES

Our investigation of the existing literature as well as the Broadcom Edge-Core AS5712-54X concluded that albeit needed for ISP use cases, there is no virtualization approach available for renewable resources on the control path of SDN data elements. Many resources in the resource topology of the Edge-Core AS5712-54X requires this type of virtualizer: the memory interfaces of flow tables, group tables, meter tables, statistic counters, as well as the management system. Therefore, we provide the first virtualizer for renewable control path resources. We achieve this by providing the required resource visibility and resource controllability to the controller by using existing OpenFlow mechanisms on the example of adding flow table entries. This OpenFlow primitive operates on a TCAM-based hardware match table resource, which is the most demanding for virtualization. The proposed virtualizer is therefore also applicable to the other, less demanding resources.

Literature shows that inserting entries into flow tables specifically often yields a slow and unpredictable rate on hardware switches [Kuz+18]. The resource causing this behavior has been identified, as discussed in Section 4.3, to be the TCAM matching memory. When multiple control plane applications access the same flow table at the same time, the slow speed of the insertion of flow entries quickly makes this operation a bottleneck. Furthermore, the unpredictability of the rate means that the virtualized resource will exhibit unpredictable behavior as well. The proposed virtualizer fulfills the requirements of reliable control plane applications in ISP networks: prioritization and fairness as discussed in Section 4.1.1. Prioritization is required to ensure that critical control plane applications such as unicast routing are never interfered with by other, less important services. Fairness is needed to ensure that control plane applications with the same priority get the same services.

The goal of the approach presented in this section is to demonstrate that a virtualizer operating on the control plane can gain visibility and control over the resource in the data plane as well as provide prioritization and fairness.

An analysis of the resource path to the TCAM-based flow table through the resource topology of a Broadcom Trident II-based OpenFlow device is provided in Section 4.4.1. The design for a virtualization approach for adding flow entries into TCAM-based flow tables with prioritization as well as our prototypical implementation is described and is presented in Section 4.4.2. The evaluation design and the testbed setup are discussed in Section 4.4.3. Finally, the evaluation results are presented and analyzed in Section 4.4.4.

4.4.1 *OpenFlow Flow Entry Addition Analysis*

The resource path used when adding entries to OpenFlow flow table is highlighted on the background of the Edge-Core AS5712-54X's resource topology in Figure 4.15. The bottleneck in this resource path has been established in literature to be the addition of flow entries into TCAM memory. This analysis starts with an in-depth analysis of TCAM memory and then follows the resource path as well as the overlaying OpenFlow mechanisms to the OpenFlow controller.

The characteristics of TCAM-based memory have been discussed in Section 4.3.3. From the perspective of the OpenFlow controller, the processing time for adding entries

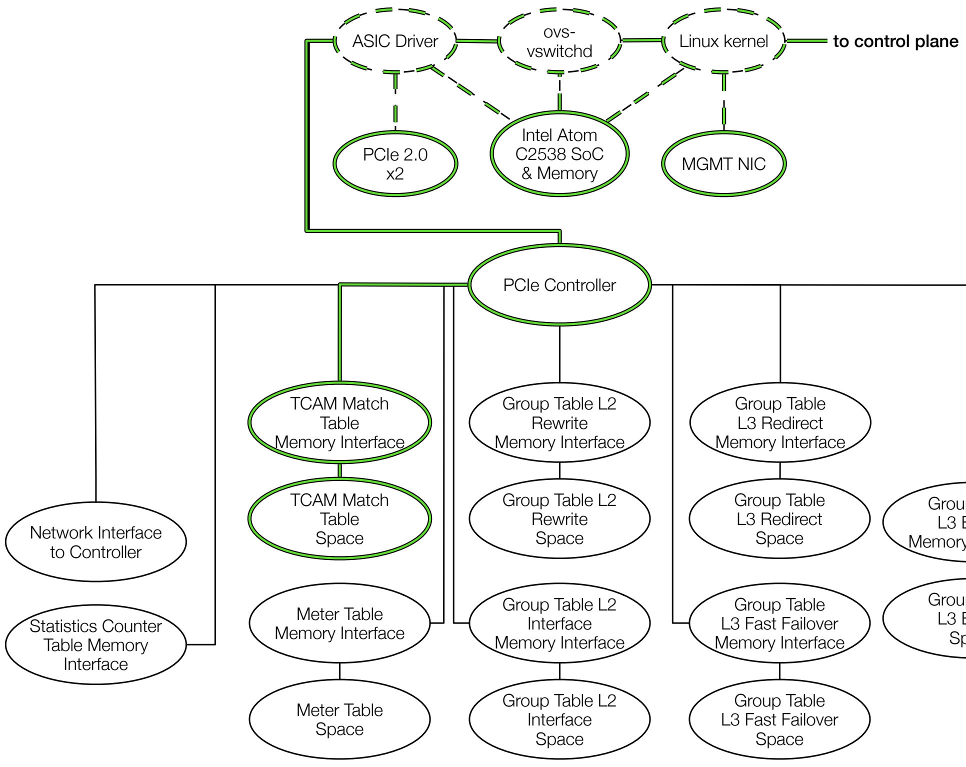

ASIC Driver
ovs-vswitchd
Linux kernel
to control plane
PCIe 2.0 x2
Intel Atom C2538 SoC & Memory
MGMT NIC
PCIe Controller
TCAM Match Table Memory Interface
TCAM Match Table Space
Group Table L2 Rewrite Memory Interface
Group Table L2 Rewrite Space
Group Table L3 Redirect Memory Interface
Group Table L3 Redirect Space
Group Table L3 ECMP Memory Interface
Group Table L3 ECMP Space
Network Interface to Controller
Statistics Counter Table Memory Interface
Meter Table Memory Interface
Meter Table Space
Group Table L2 Interface Memory Interface
Group Table L2 Interface Space
Group Table L3 Fast Failover Memory Interface
Group Table L3 Fast Failover Space

Figure 4.15: The resource topology of the Edge-Core AS5712-54X with the resource path for OpenFlow flow_mod messages that add flow entries highlighted.

is not deterministic and varies according to the priority of existing and new flow entries. Therefore, the TCAM-based flow table memory interface is expected to be the bottleneck resource.

The next resource on the path to the controller is the ASIC's PCIe controller, followed by the PCIe controller of the data plane CPU. Then, the ASIC driver takes over, followed by the OpenFlow agent and the network process that receives packets from the OpenFlow controller. Based on the existing literature, none of these resources or components is expected to be slower in processing than the TCAM memory interface. Therefore, we do not expect any of these resources to be a bottleneck. The next step in this analysis is the mapping of the resources to the corresponding OpenFlow messages.

Flow tables are the central abstraction in OpenFlow. Therefore, flow table updates are a crucial feature for control plane applications to control the data plane. Flow table modifications, in general, are implemented by the OpenFlow primitive flow_mod. The flow_mod message provides fields for specifying all aspects of a flow entry. The intention of a flow_mod message, whether entries should be added, modified, or deleted is encoded in the *command* field. The data plane evaluates flow_mod messages by applying its *command* to the specified flow table. This is done by first selecting the flow entries that are affected by the flow_mod message and the applying then specified *command* to them. Flow entries are selected if their match fields match the same packets or a subset of the packets that are matched by the match fields that are specified in the flow_mod message. This means, for example, that a flow_mod message with a wildcard matcher for all packet header fields will match all existing flow entries. While this approach makes this flow_mod message very flexible and generic, it makes the analysis, which resources are affected by a given flow_mod message, challenging. Different *commands* affect different resources as we will explain later in the section. Furthermore, for every flow_mod message, the number of flow entries affected by it must be determined for each message.

For the *add command*, which we will analyze in detail, the investigation of the number of affected flow entries is simpler than for the *modify* and *delete* command. While the latter can potentially affect all flow entries in a table, the *add* command may only affect a single entry with identical match fields. If such an entry exists, it is replaced; if it does not, the specified entry is added to the table.

This behavior of slow and unpredictable flow entry addition times has been investigated in the literature [Kuz+18]. However, no approach exists on investigating the characteristics of interference between multiple control plane applications OpenFlow flow table updates.

4.4.2 *Virtualizer Design*

The goal of the virtualizer is to provide prioritization and fairness for adding entries to flow tables with unpredictable addition performance, such as TCAM-based flow tables. Furthermore, the virtualizer operates on a distributed system, with the resource being located on a data plane element and being controlled by an SDN controller on a different device.

The unpredictable flow entry addition rate limits the design space for these goals. Well-proven approaches that involve modeling a static flow addition rate suggested, e.g., by Bozakov and Rizk [BR13] and Blenk et al. [BBK15] for similar problems cannot be used to provide prioritization in this scenario. With performance modeling not being applicable, we conclude that a feedback mechanism is required for the OpenFlow switch to provide information to the controller, which flow entries have been already installed. This leaves queueing approaches as a well understood and matching design approach to tackling the issue.

An overview of an SDN message queueing system and its main components are given in Figure 4.16. The two main components are the queue manager and the scheduling algorithm. The queue manager decides in which queue a new work item should be placed. The scheduler decides which queue should be serviced next and how much service it should get.

Before discussing the design of the solution for the distributed queueing system, we characterize the queueing and scheduling problem at hand:

- One bottleneck resource: the TCAM
- No preemption: once the flow entry addition process has been started, it cannot be interrupted
- Variable execution times: the time it takes to add flow entries is not static
- The bottleneck resource control operates on the OpenFlow switch and the scheduler operates on the OpenFlow controller
- Strict priority scheduling and fairness for equal priority is required
- Flow entry message must not be lost or dropped

As depicted in Figure 4.17, one or more flow tables are available in the ASIC. For this investigation, all resources that are not relevant are left out to ensure readability. The controller must keep track of the flow entries that have been sent to the OpenFlow switch and wait for each of them to be reported as installed by the switch. Using a queue management approach, the controller can schedule the application requests either for fairness or prioritization. Each hardware flow table is represented by a set of queues and a scheduler. The controller analyzes each flow entry addition requests from the applications to determine which hardware flow table they will be added to. Then, according to the priority of the application, the request is added to one of the queues of the corresponding flow table. For each table, a scheduler then decides which queue to service next, depending on the feedback information from the switch. The approach is depicted in Figure 4.17. The priorities are depicted in red for high priority, orange for mid priority, and green for low priority. The applications themselves can assign different priorities to their messages. These are combined with the application's priority to determine the actual per-flow table priority.

The most straightforward approach is to order the OpenFlow messages by their prioritization and send them to the data plane device that processes them in a first in, first

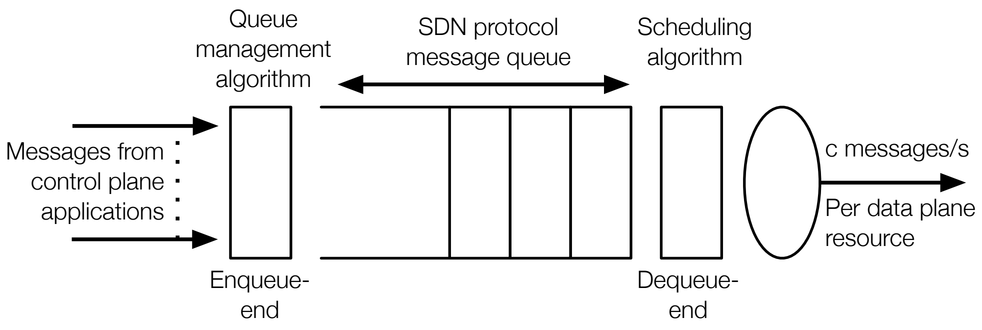
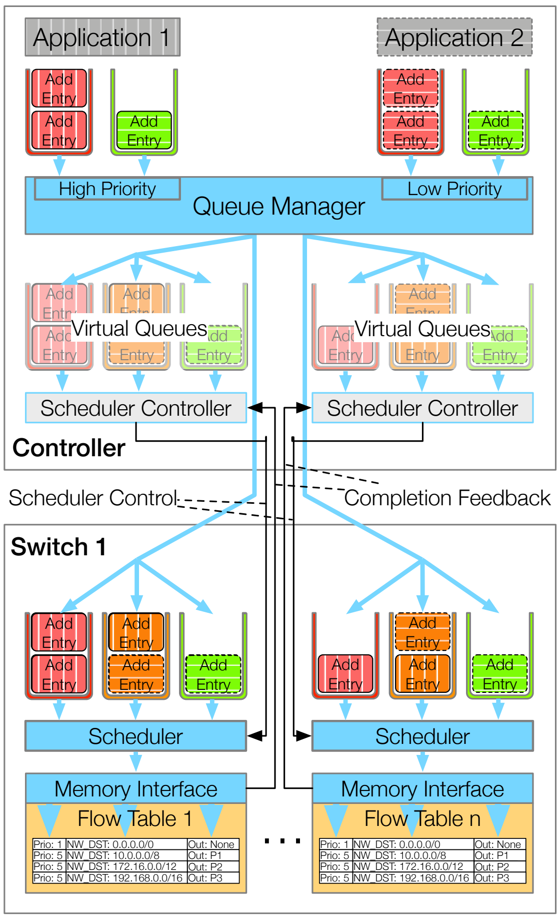

Queue management algorithm
SDN protocol message queue
Scheduling algorithm
Messages from control plane applications
c messages/s
Per data plane resource
Enqueue-end
Dequeue-end

Figure 4.16: Overview over an SDN message queueing system and its terminology (adapted from [AHA16]).



Prio: 1	NW_DST: 0.0.0.0/0	Out: None
Prio: 5	NW_DST: 10.0.0.0/8	Out: P1
Prio: 5	NW_DST: 172.16.0.0/12	Out: P2
Prio: 5	NW_DST: 192.168.0.0/16	Out: P3

Prio: 1	NW_DST: 0.0.0.0/0	Out: None
Prio: 5	NW_DST: 10.0.0.0/8	Out: P1
Prio: 5	NW_DST: 172.16.0.0/12	Out: P2
Prio: 5	NW_DST: 192.168.0.0/16	Out: P3

Figure 4.17: Design model for flow_mod virtualization with prioritization (adapted from [Vil18]).

out (FIFO) order. Unfortunately, the OpenFlow standard specifies that the processing of OpenFlow messages does not have to occur in FIFO order; „hence, controllers should not depend on a specific processing order“. Using a priority queue on the data plane device and tagging OpenFlow messages accordingly provides prioritization but provides the controller little control and feedback over the process. If the waiting times caused by the queueing become too long, the controller has no option to remove requests from low priority queues and provide information to the application. The same is true if requests are dropped from the queue, providing information on that to the applications becomes difficult for the controller. Another approach would be to implement a Transmission Control Protocol (TCP)-like end-to-end flow control mechanism with prioritization between the controller and each resource. While this approach is promising as well, it requires a significant redesign of the OpenFlow agent on the data plane devices. However, this is not possible with existing OpenFlow agents of available hardware devices, because of their proprietary nature. Furthermore, the TCP connection used to transport the OpenFlow protocol would have to be replaced by one that supports multiple independent streams, such as Stream Control Transmission Protocol (SCTP).

One problem is that OpenFlow does not provide a primitive to monitor or control the hardware flow tables in the ASIC directly from the controller. Instead, the OpenFlow model presents the available matching memory as a pipeline of flow tables, independent of how the hardware looks like. Furthermore, OpenFlow does not provide any primitive to get information on the processing of OpenFlow messages inside of the switch. Two OpenFlow primitives exist to get information on the state of the processing of OpenFlow message:

- barriers, available from OpenFlow version 0.9
- bundles, available from OpenFlow version 1.4

The barrier command should ensure that all OpenFlow messages that the switch received before the barrier_request message have been completely processed before the barrier_reply is sent. Unfortunately, since this command includes all OpenFlow messages, it cannot be used to track flow entry addition operations for a specific hardware flow table only. Therefore, it cannot be used. The flow bundle feature, however, is more interesting. It was designed to improve the synchronization of flow table modifications by providing a primitive to defined groups of OpenFlow messages that are executed at the same time. Bundles are opened by sending the corresponding message including an id to the switch. Then, the controller can add arbitrary OpenFlow messages to the bundle until it decides to close it. Another advantage is that the messages are already validated when they are added to the bundle, which reduces the processing time later. After that point in time, the bundle can be committed or discarded. If it is committed, the switch will reply with a bundle_commit_reply message that indicates if all messages in the bundle have been committed successfully.

The ability to open multiple flow bundles in parallel, commit them independently from each other, and get an acknowledgment for each of them means that this feature could be used to implement queuing control on an OpenFlow switch per hardware flow table. The disadvantage, however, is that when flow bundles are used to implement queuing for flow entry additions, control plane applications and OpenFlow controllers that use the flow bundles feature already for other purposes cannot be supported. Furthermore, the order of the processing of the OpenFlow message processing could be changed by the separation of flow entry additions from the other messages. However, the goal of this design is to show the usefulness and possibility of control path virtualization in SDN protocols, not to fix all shortcomings of OpenFlow. Since using flow bundles is the only available way to implement this feature on an actual hardware switch, we will proceed with this design.

A priority round robin scheduler is used on the OpenFlow controller inspired by the work of Tsao and Lin [TL01]. The quantum of the scheduler, which is the part of work assigned to each queue per time interval, e.g., the number of transmitted bytes for packet schedulers is defined differently for this use case. As discussed before, the processing times per flow entry addition are dynamic. Therefore, we need to share the time used per application instead of the number of executed flow entry additions. Therefore, we assign time slots directly to the queues, by specifying the quantum in milliseconds. Since preemption is not possible the quantum size has a significant influence on the granularity of control. Once the assigned service quantum is consumed by a queue the next queue with the same priority is selected. The scheduler keeps at least one flow bundle open per hardware flow table on the OpenFlow switch. New flow_mod messages are, after determining their destination hardware flow table, placed in a queue on the controller and added to one of the corresponding flow bundles on the switch by the queue management algorithm. The scheduler selects the next queue to service, commits the next bundle, and determines, based on the time slice used by the queue, which queue should be serviced next. Furthermore, the scheduler must wait for the bundle_commit_reply message before updating the time slice calculation.

Queues are selected by their priority to implement strict prioritization. This means that the set of queues that contain messages with the highest priority are always served until they are empty. Only then queues with lower priority are served as well. By ensuring that higher application always take precedence as required in ISP networks, this approach can lead to starvation of lower priority control plane application. The effects of starvation must be handled by each application individually, by implementing control path bottleneck mitigation approaches as discussed in Chapter 5.

The flow entry addition scheduler was prototypically implemented using the Floodlight OpenFlow controller[9]. The focus of the prototype is the investigation of the system's behavior in an overload event of the TCAM match table memory interface resource. The goal of the evaluation is to show if an SDN protocol feature like the flow bundle can provide the required functionality to implement a virtualizer. Only the parts need to support this use case are implemented.

The design of the prototypical implementation of the system reflects a queueing system design, except that it is event-driven due to the nature of OpenFlow controllers. The two main entities in the prototype implementation, like in most queueing systems, are the queue manager and the scheduler. The queue manager opens and closes flow bundle as well as assigns messages to queues and bundles. The scheduler is responsible for selecting the queue that should be serviced next and initiate the servicing by committing flow bundles. When a control plane application requests a flow entry addition, the queue manager checks if an open flow bundle is available for the addressed switch and flow table. In this experiment, flow bundles are created proactively when the switch connects to the controller to ensure that the flow bundle creation does not affect the scheduler performance. If an open flow bundle is available, the flow is added to that bundle. If the selected bundle has reached its maximum size, it is closed, and the scheduler is called.

9 Project Floodlight. *Floodlight OpenFlow Controller*. Accessed: 2018-09-12. url: `http://www.projectfloodlight.org/`.

The scheduler calculates the queue that should be serviced next and commits its flow bundle. Once the switch reports back that the bundle has been committed successfully, the corresponding service time information is collected, and the scheduler is called again.

Though this approach, the queued messages for each resource are kept on the data plane device while the queue with references to these messages is kept on the controller. Thereby, the controller has full knowledge of the queue lengths. The scheduler is kept on the controller and can modify, rearrange, or discard bundles at will. Furthermore, errors that happen while committing bundles are reported by corresponding OpenFlow protocol messages. Thereby, the requirements for virtualizing resources are met. Resource controllability is provided through controlling and committing bundles by the control plane. The information required to gain resource visibility: resource utilization, resource saturation, and resource errors are provided except for resource utilization. Resource saturation information is provided by the number of the outstanding flow addition entries. Resource errors are provided through the OpenFlow protocol. However, gathering detailed information on the resource utilization of the TCAM match table memory interface resource requires a modification of the ASIC driver and the corresponding OpenFlow facilities to transport this information to the controller. Since we do not have access to the proprietary ASIC drivers, we rely on an estimation. The utilization of the resource can be estimated to be either 0% or 100% determined by the available saturation information.

4.4.3 *Evaluation Design and Testbed*

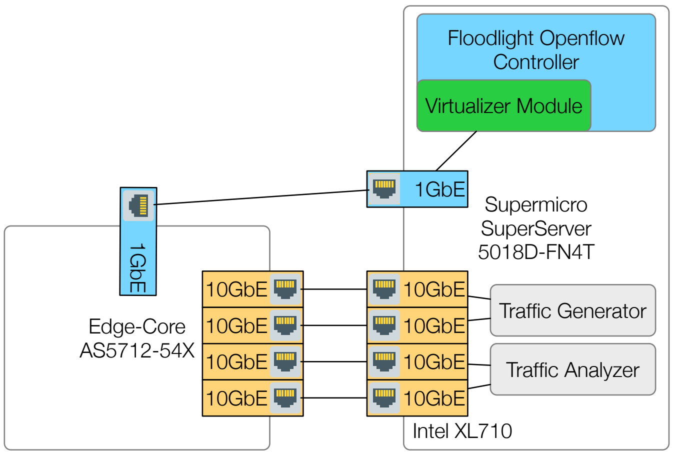


Figure 4.18: Testbed overview of the flow_mod virtualization evaluation (adapted from [Vil18]).

The testbed setup is depicted in Figure 4.18. An Edge-Core AS5712-54X operating Pica8 PicOS 2.8 is the investigated OpenFlow switch under test. One server is used to operate the OpenFlow controller, as well as generate and receive traffic. This configuration ensures the consistency of the time stamps, as they are all created using the same time source. The server is a Supermicro SuperServer 5018D-FN4T that has an Intel Xeon D-1541 CPU with eight cores and 64GB RAM to ensure the OpenFlow controller and the traffic generators and receivers do not create a performance bottleneck on the server. It is connected to the management interface of the switch under test using a single 1GbE link. Its connection to the data path of the switch is four 10GbE links provided by a single Intel XL710 NIC with a single 40GbE ports through a breakout cable.

The prototype implementation supports the evaluation by implementing an event-driven flow entry addition generator. A packet generator on the measurement server sends packets to the OpenFlow switch. For each IP address in these packets, for which no flow entry exists yet, a packet_in message is created on the switch and sent to the controller. There, the controller extracts the destination IP address from the incoming packet and uses it to create a flow entry addition request. This approach ensures that the rate at which the flow_mod messages are sent do not create a performance bottleneck on the management system CPU resource but in the TCAM match table memory interface resource. Thereby, application load is generated directly in Floodlight by simulating two different control plane applications.

Literature has shown that descending flow entry priority represents the worst-case performance [He+15; Laz+14]. Hence, the flow entry addition messages are given a continuously descending OpenFlow flow entry priority. Once an entry exists for a given destination IP address, no further packet_in messages are created and the packet is forwarded in the data path of the switch. This point in time signifies the successful installation of the flow entry and is recorded on the measurement server by capturing the forwarded packet. As stated in the data plane analysis, the confirmation messages sent by the Edge-Core AS5712-54X that the bundle was committed successfully are used for scheduling but are in general not deemed accurate to reflect the actual state of the ASIC.

To show the usefulness of the approach, we investigate the prototypical implementation of the system in a minimal configuration with two control plane application and one OpenFlow switch. The goal is to show that strict prioritization works as expected and that the fairness is given for applications with the same priority. Furthermore, we investigate the effect of two different quantum sizes for the scheduler. The analyzed metric is the time it takes from sending the flow entry addition message to the OpenFlow switch to the time when the new flow entry affects the network traffic passing it. For strict prioritization, we expect this time to be as small as possible for the high priority application. The proposed system is compared to installing the flow entries without flow bundles because it is the prevalent method of adding flow entries.

Initial experiments were conducted to determine the parameter space suitable for the experiments. Specifically, we investigated the flow entry addition behavior when filling the complete flow table at the same time. It has been described by, e.g., Kuzniar et al. [Kuz+18] that some Broadcom-based OpenFlow switches such as the Dell 8132F

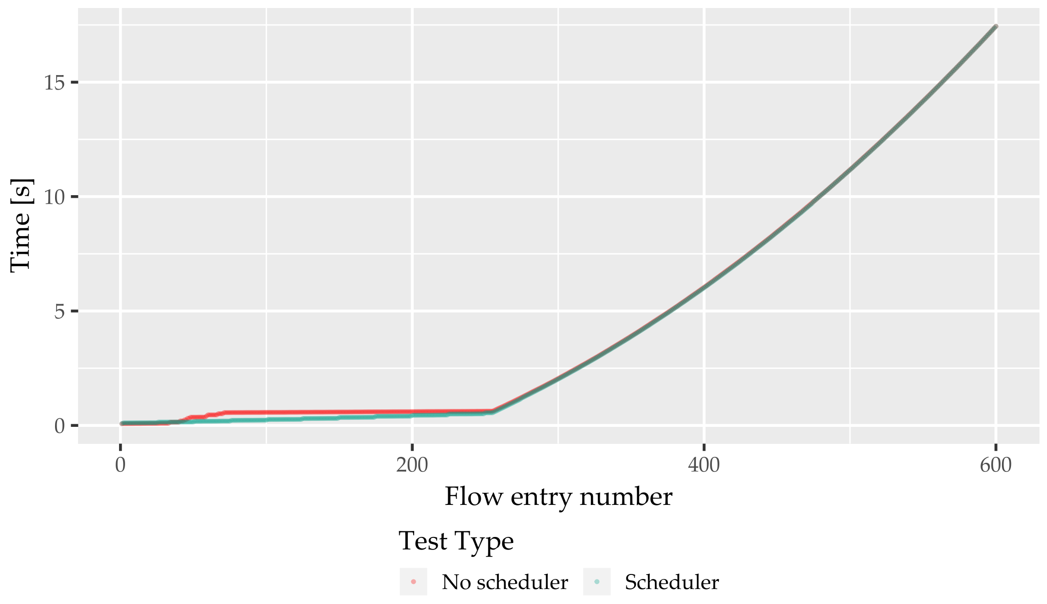


Figure 4.19: The mean flow entry addition completion time for the Edge-Core AS5712-54X when filling the complete flow table (adapted from [Vil18]).

exhibit a performance anomaly when installing more than about 200 flow entries. While the Edge-Core AS5712-54X based on the Trident II ASIC and not the Trident+ as the Dell switch mentioned earlier, we observe a similar behavior. The results of filling the complete flow table is depicted in Figure 4.19. The means of the installation time per flow entries of 10 repetitions of the experiment are depicted. The depiction clearly shows a sharp increase of the completion time when more than ~250 flow entries are installed consecutively. To investigate this phenomenon in more detail, we visualize the sudden drop in performance by plotting the mean installation time per flow entry is in Figure 4.20. The flow installation performance drop happens between 245 and 260 flow entry installations. The performance drop occurs at different but similar point as in the paper of Kuzniar et al. [Kuz+18] where it appears around 210 flow entries. Since the PicOS software is closed source, a more detailed investigation is difficult, if possible at all. We suspect that the behavior is caused by the fact that both the PicOS software and the software of the Dell 8132F might rely on the same driver software to communicate with the ASIC that is provided by Broadcom. In any case, we consider this behavior an anomaly. Furthermore, we cannot identify the root cause if it, which could mean that, e.g., the flow bundle mechanism is affect by this anomaly as well. Therefore, the anomaly should not influence the investigation of the scheduler. Hence, we design the experiments to stay below the number of flows that cause this anomaly.

The load parameters used are listed in Table 4.6. As stated before, to remove the interference of the creation of flow bundles they are created proactively on the switch.

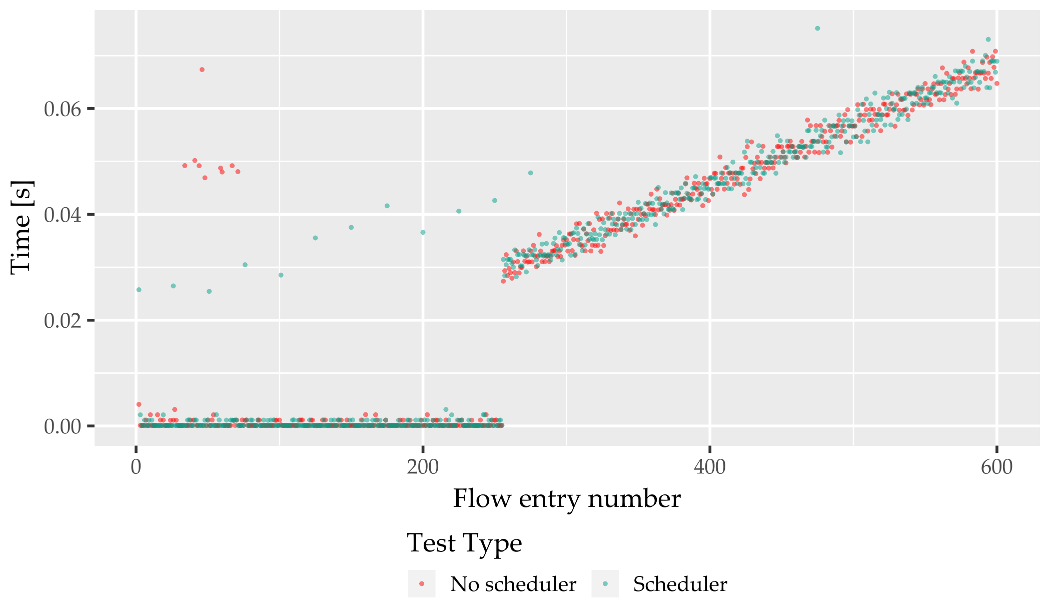


Figure 4.20: The mean flow entry installation time for the Edge-Core AS5712-54X when filling the complete flow table (adapted from [Vil18]).

Table 4.6: Evaluation parameters and values (adapted from [Vil18]).

Parameter	Values	Description
Repetitions	10	Number of repetitions per experiment
Scheduler	with/without	Using the proposed scheduler or not
Quantum	{ 1ms, 10ms }	Quantum sizes used by the scheduler
Number of flow entries	50	Total number of flow entries that are added during the experiment per control plane application
App number	2	Number of simulated control plane applications
App priorities	{ Same, High/Low }	Priorities of the simulated control plane applications

Since the Edge-Core AS5712-54X is limited a maximum of ten open flow bundles at a time[10] and we use at maximum 10 flow entry additions per flow bundle, a total of 100 flow entries can be installed on the switch–which is well below the number of flows that cause the performance anomaly. This means that for two applications, 50 flow entry additions are executed. The investigated quantum sizes are one and ten milliseconds to

10 Pica8. *PicOS Open vSwitch Command Reference, PicOS 2.8*. Version 1. Jan. 2017.

ensure the necessary freedom of control for the scheduler according to the time spans that are observed in initial experimentation.

4.4.4 *Evaluation Results*

The results of the evaluation are discussed in this section. To investigate the characteristics of the scheduler, we use the default flow installation approach with the scheduler and flow bundle as the baseline. Note that in all the following figures the baseline experiment results are depicted in red. The baseline is compared to two experiments with the scheduler: one with a quantum size of 1ms for both applications and one with a quantum size of 1ms for the low priority control plane application and 10ms for the high priority application. Experiments, where both applications use the same quantum size, are depicted in green, experiments, where the applications use different quantum sizes, are depicted yellow.

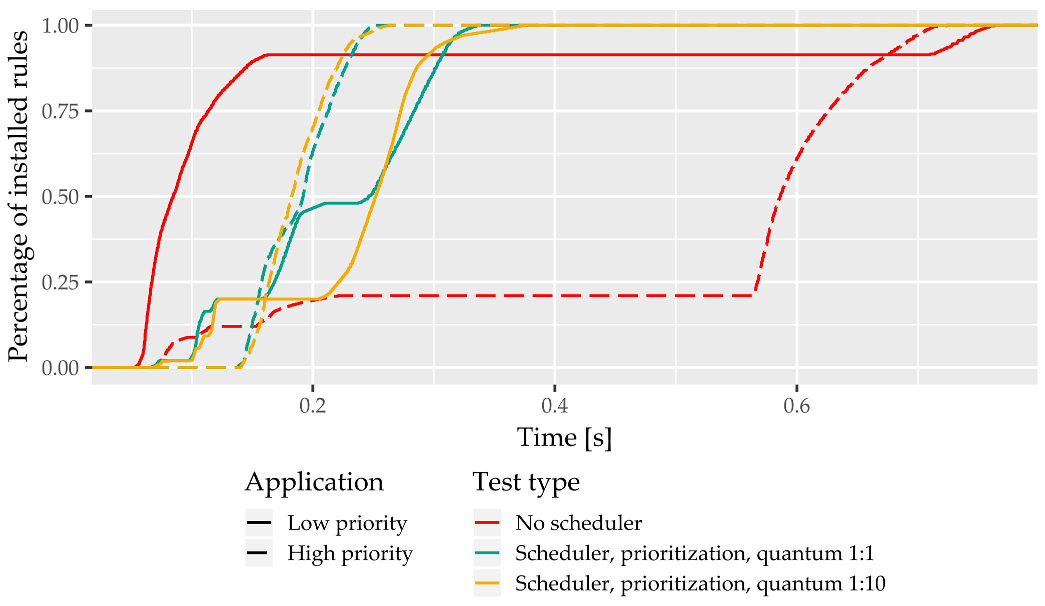


Figure 4.21: Flow entry addition time with and without the scheduler for the prioritization use case (adapted from [Vil18]).

In Figures 4.21 and 4.22, the effect of the scheduler for the prioritization use case is depicted. Note that the low priority application starts to send flow entry addition messages first; the high priority application starts shortly after. This approach reflects a worst case situation, where a low priority application could potentially stall flow entry additions of a high priority application that are required to prevent packet loss.

In the depiction of the cumulative distribution function of all ten repetitions of the experiment in Figure 4.21 clearly show that independent of the quantum parameters, the

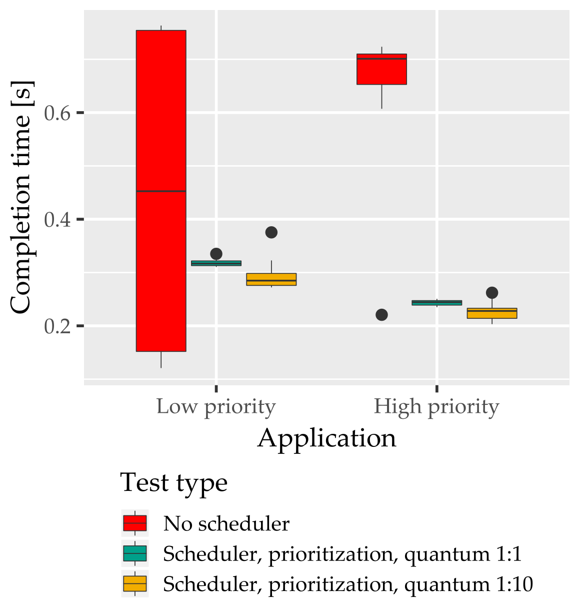


Figure 4.22: Flow entry addition completion time with and without the scheduler for the prioritization use case (adapted from [Vil18]).

scheduler completes the flow entry addition for both applications in about half of the time than the experiment without the scheduler. The reduction of the total flow entry addition time is an unexpected but positive effect. Due to the overhead introduced by the flow bundle mechanism and the scheduler code on the OpenFlow controller we expected a small completion time increase when introducing the scheduler. Since the differences between the two quantum size configurations are small, we assume that this effect is caused by the flow bundle feature. While not required by the OpenFlow specification, it seems that the PicOS software uses the knowledge that the flows are added in a batch to optimize the addition process. We deduce that instead of writing the TCAM-based flow table entry by entry, a bulk mechanism is used to write all flow entry in one go. Not only can this be done more efficiently, but the reordering of flow entries that might be necessary do not have to be applied one-by-one. Instead, we suspect that for all new flow entries in a bundle the new TCAM table is calculated and then written once. This is consistent with the flow entry addition completion times for the experiment depicted in Figure 4.22.

The variability in the results of the experiments without the scheduler is more significant than the ones with the scheduler. Especially the completion times of the low priority application differs significantly between the repetitions. We interpret this result in that the flow entry controller of PicOS does not use a strict FIFO queue for scheduling flow updates. Instead, sometimes flow entry addition requests that were sent later are installed before the older ones have been completed. We suspect that this behavior is caused by the software design of the OpenFlow agent. This inconsistent behavior of the OpenFlow agent shows that prioritization and queueing features need to be controlled by the SDN controller to ensure consistent behavior. Note that the flow entry addition completion time for the low priority application without the scheduler is much lower until about 90% of the entries than the one with the scheduler. At that point, the flow entry addition requests of the other application take over and stall the first application for more than 0.4s. The fact that PicOS does not seem to use a FIFO queue to process flow entry additions is surprising. In the use case investigated, it results in unpredictable completion times with high variability. Furthermore, we cannot provide an interpretation of what might have been the motivation for this design decision.

When using the scheduler with flow bundles the variability is much smaller, the 25% and 75% quartiles, represented by the hinges are consistently small. The consistency for the experiments with the 1ms quantum is even higher than the one with the mixed quantum durations. However, this improvement comes at the price of slightly higher completion times that when mixed quantum durations are used. The main result, however, is that in both quantum configurations, the high priority application completes the flow entry addition process before the low priority application. This is the case even though the low priority application starts first and can install some flow entries before the flow entry addition requests of the high-priority application arrive. These results show that the scheduler works as expected and that strict prioritization can be archived when the characteristics of the hardware are understood, and the SDN protocol provides the corresponding primitives.

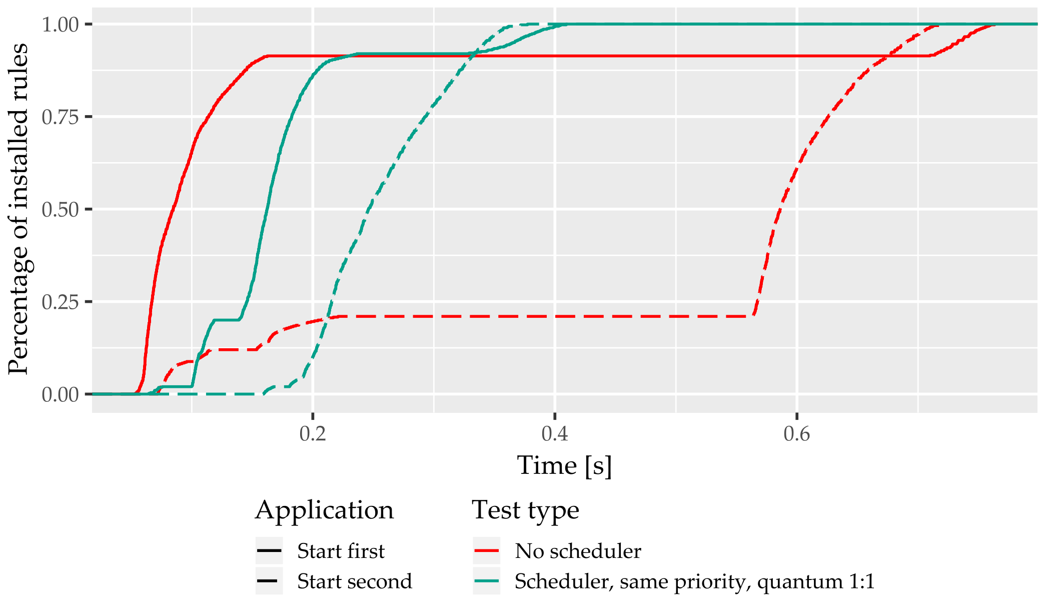


Figure 4.23: Flow entry addition time with and without the scheduler for the fairness use case (adapted from [Vil18]).

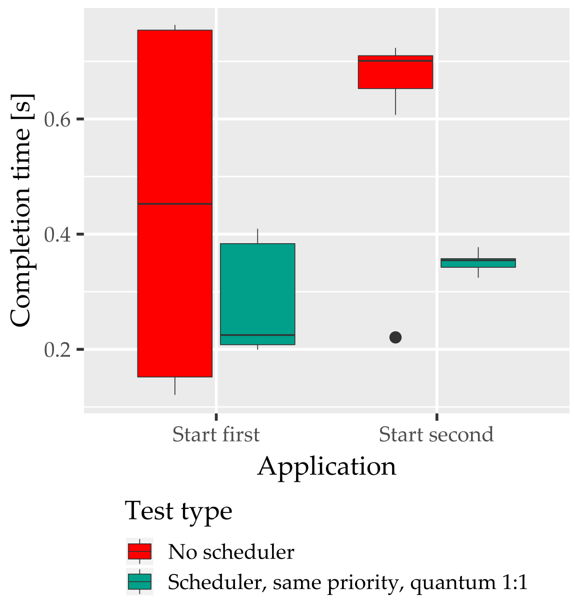


Figure 4.24: Flow entry addition completion time with and without the scheduler for the fairness use case (adapted from [Vil18]).

Besides the prioritization use case, fairness is an important aspect. We expect control plane application that are part add-on services to have the same priority. The results of the experiments without the scheduler are the same as above. There is no difference between these two experiments except the application priority, which cannot be specified without the scheduler. Therefore, the behavior that can be seen in the results without the scheduler is the same as before as can be seen in Figures 4.23 and 4.24. The behavior of the system with the scheduler shows lower variability and smaller completion times. However, the flow entry addition times for the application that starts first show an increased variability, which is similar to the behavior of the system without the scheduler. Nevertheless, the completion times for the two applications are remarkably similar, albeit not the same. Therefore, we conclude that while strict fairness could be only partly achieved in the investigated use case, the fairness could be increased.

For more details on the system implementation and extended results, please refer to the master's thesis of F. Villa-Arenas [Vil18].

4.5 DISCUSSION AND CONCLUSION

We presented an approach to discover and analyze potential performance bottlenecks in SDN data planes in this chapter. Starting at the cause of performance issues, the hardware, a new abstraction was introduced, the resource. Based on this concept, we were able to create a simple, yet powerful tool, the resource topology that SDN controllers use to reason about the performance of their data plane.

Its feasibility was shown by applying it to a representative data plane device, the Edge-Core AS5712-54X. A resource class was identified that had not been virtualized yet: reading and writing values to tables in the forwarding ASIC. Ignoring these resources means that the isolation between control plane application is not complete, which in turn can lead to uncontrolled performance bottlenecks. By presenting a virtualization approach of the flow entry addition for TCAM-based flow tables by using existing OpenFlow primitives, we closed this gap. The virtualizer clearly shows that resource-related information provided by the data plane as well as the cooperation of the control and data plane are required to implement controlled virtualization and avoid unexpected behavior of control plane applications.

Furthermore, we showed that the prioritization of applications is possible when the data plane and control plane exchange information on resources. In the presented design, an existing OpenFlow primitive, the flow bundle was used to implement this information exchange. Our evaluation shows that the approach indeed fulfills the virtualization requirements of SDN control planes. The design outperforms the prevalent approach to installing flow entries by a wide margin, in addition to providing prioritization and fairness. While the application of this approach is limited by the re-used of an existing OpenFlow primitive, the performance and usefulness of the mechanism were demonstrated on a state-of-the-art SDN switch. Furthermore, the analysis clearly showed the need for this kind of mechanism in future SDN protocols, such as P4Runtime [P4Runtime].

While we presented the first systematic approach to discovering data plane resources and proved its usefulness, we also discovered areas for further improvement. The analysis works best when the hardware resources of a device are known, which is not the case for many packet processing ASICs today. Furthermore, the goal of SDN protocols is abstraction, i.e., the reduction of information. The bottleneck analysis introduces a lot of new information requirements to SDN protocols. This information, however, is difficult to add to SDN protocols like OpenFlow that were designed with a focus of functional abstraction. We, therefore, suggest that next generation protocols, such as P4Runtime [P4Runtime] should include resource information in their design. Our findings support the idea of Ousterhout et al. [Ous+17] that performance clarity should be a first-class design principle for both hardware and software designs to achieve efficient, predictable, and controllable SDN control planes in future ISP networks.

5 DESIGNING RELIABLE CONTROL PLANE APPLICATIONS FOR VIRTUALIZED SDN DATA PLANES

In this chapter, we discuss our answers to Research Questions 2.1 and 2.2 that are stated in Chapter 1. From the perspective of control plane applications, virtualized SDN data planes differ from non-virtualized data planes in one significant aspect: virtualized control planes show a higher variability in resource availability and control path bottlenecks caused by contention.

We investigate the nature of control path bottlenecks by classifying them and analyzing how they are perceived by the SDN control plane. Furthermore, we investigate the design space for control plane applications to react to these events to ensure reliable operations. The goals of this investigation twofold: the first goal is to demonstrate that applications can mitigate the effects of performance events to improve their reliability. The second goal is to analyze what kind of information the control plane applications require from the controller to execute the mitigation approaches.

The investigation is conducted in the context of two representative control plane applications: network function chaining and multicasting. Network function chaining is a crucial service that enables NFV and focusses on the data center parts of ISP networks. Multicasting applies to all parts of ISPs networks. It provides the distribution of packet streams, e.g., videos, between the core network, the ISP's customers, and the rest of the Internet.

We start the investigation by analyzing control path bottleneck events as well as the design space for mitigating them in Section 5.1. The scenarios and use cases that motivate the two representative control plane applications, network function chaining, and SDM, are presented in Section 5.2. A design for a network function chaining system that can mitigate local control path bottlenecks is introduced in Section 5.3. Our proposal of SDM-based design that can mitigate global resource shortages in the control paths of the data plane is discussed in Section 5.4.

The investigation in this chapter partially relies on input from three papers [Ble+14; Ble+15a; Ble+15b] and three supervised student theses by S. Bleidner, T. Volk, and P. Welzel [Ble15; Vol14; Wel16].

5.1 DESIGN SPACE ANALYSIS

To discuss the changing availability of data plane resources, we introduce the term resource event.

Definition 10: Resource event
An event that is caused by a resource shortage in the data plane that affects the operations of control plane applications.

We focus on control path resources in this investigation, but the definitions apply to data path resources as well. At the core of resource events are information gathered from single elements of the resource topology in the data plane. These events are termed local resource event.

Definition 11: Local resource event
A resource event that affects a single data plane element of a network domain.

If these events directly affect control plane applications, they must react accordingly. If a local event does not directly affect applications, it is possible that it contributes to the formation of a global resource event:

Definition 12: Global resource event
A resource event that affects a whole area of the data plane of a network domain.

In both cases, the controller or the application must identify such an event and initiate a mitigation process if required. To connect the data path and control path resources, the resource topologys of all data planed elements are interconnected using a topology discovery process, which is available as a standard feature in today's SDN controllers. The resulting data plane resource topology contains all resources in the controlled network. The view of the control plane application on the network topology can be restricted or modified by the controller. Therefore, the same process that is used to restrict the view of control plane applications on the network topology is applied to the data plane resource topology to determine all control path resources in the view of a given application.

This virtualized view on the network influences the emergence of events since the available resources are limited to this view. Therefore, both local and global events do not necessarily affect the same control plane applications even though they might use some of the same data plane elements. One of the goals of this investigation is to determine which entity, the controller or the application should trigger an event. Furthermore, it will be analyzed what kind of information is required from the controller and the application to trigger the event and to decide how to react to it.

To operate reliably, control plane applications need to be able to mitigate the effects of resource events. In general, applications should consume as little resources as possible to avoid being affected by a resource event. Therefore, all applications should be optimized for their data plane resource consumption.

If a resource efficient control plane application is nevertheless affected by a resource event, it must decide how to react to the issue. If a specific resource is unexpectedly not available to an application, the application's resource consumption must be modified, or

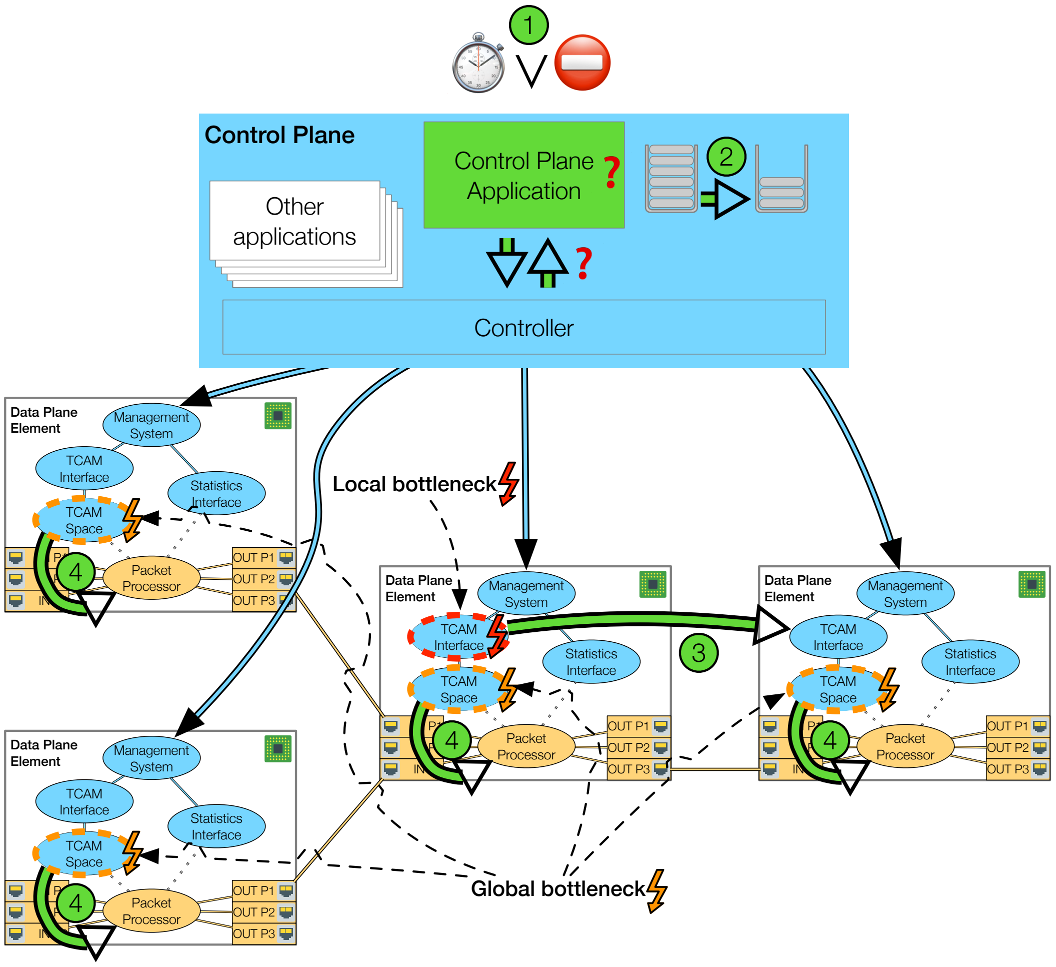


Figure 5.1: Overview of resource events and corresponding mitigation strategies.

the whole operation has to be canceled. Depending on the resource and the event, the modification of the resource consumption can take place in one or more dimensions: temporal, quantitative, spatial, and type as depicted in Figure 5.1. A temporal resource modification means that the time of the resource consumption is postponed as denoted in the figure by ①. A quantitative modification means that the amount of resources to be consumed is reduced, e.g., by compressing the resource request ②. A spatial resource modification means that the resource is not used in its originally indented location, e.g., a specific data plane element, but in a different one ③. Finally, the bottlenecked resource can be replaced by a different resource type ④. These four dimensions specify the complete design space for reacting to resource events. The approaches on how to implement these resource request modification operations can vary between use cases and applications.

We focus on the primary resource events in this thesis. Multiple applications can react to a local resource event in the same manner, e.g., by a spatial modification of the resource request and selecting the same new location to conduct the planned operation. Thereby a new resource event in the new location can be created. Coordination between the application's mitigation strategies is required to prevent cascading resource events through the network. We denote the initial resource event that was not created by another resource event, as a primary resource event. Resource events that are created as the effect of another resource event are termed secondary resource events. We consider secondary resource events future work because the design space and effects of the handling of primary resource events must be understood first.

5.2 SCENARIOS AND USE CASES

The control plane application use cases analyzed in this chapter to investigate approaches for mitigating control path performance events are selected to be representative for ISP networks. Therefore, they include a low and a high priority application, and cover, between them, all relevant areas of ISP networks. To that end, two control-plane applications are investigated: network function chaining and multicasting.

Multicasting is a control plane application that shows great promise for the delivery of live-streaming media [Rüc16]. Multicasting applications interact with most parts of an ISP network, with a focus on the core and edge network. While multicasting is useful, it is an add-on service that is expected to operate at a low priority. Network function chaining is used in data centers in the context of NFV to create packet processing graphs from VNFs. The application areas for our investigation are edge data centers that are proposed for ISPs, e.g., by Peterson et al. and their CORD approach [Pet+16]. Network functions chaining is crucial for NFV, and the corresponding application is, therefore, expected to operate at high priority. In combination, both applications are involved all relevant area of ISP networks and include a low and a high priority application. Network function chaining is introduced in Section 5.2.1 and SDM in Section 5.2.2.

5.2.1 *Network Function Chaining*

To achieve consistent terminology in this thesis and avoid confusion we refer to the concept discussed in this section as network function chaining. The term network service chaining is often used in literature to denote the same concept. However, it predates the widespread adoption of NFV in academia, and was, e.g., used in our original publications [Ble+14; Ble+15a]. We use the newer NFV terminology to discuss the relevant components. The terminology is oriented on the terminology of the ETSI [ETSI18]. It is adapted for readability in this context and restricted to the relevant terms.

Network services can be created through the combination of VNFs. To achieve that, network traffic has to be identified at the edges of the data center to become subject of a network service. Then, the traffic is forwarded through a chain of VNF instances to implement the service. In this context, the traffic routing between the VNF instances does not rely on traditional routing or switching. Instead, depending on the network service's requirements for network traffic, it is forwarded in arbitrary patterns at the discretion of the network service designer. This traffic forwarding scheme is called network function chaining.

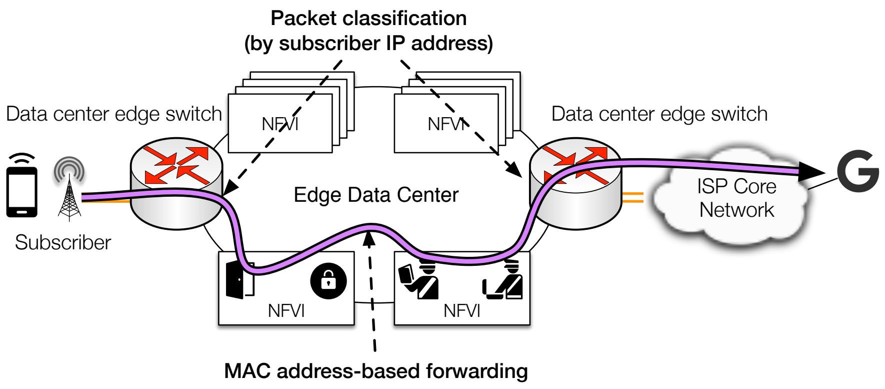


Figure 5.2: Network function chaining scenario (adapted from [Ble+14]).

An example of an edge data center that uses NFV is depicted in Figure 5.2. The ISP in the depicted example provides a mobile Internet access service to its subscribers. The service includes, besides a subscriber gateway function, a firewall and a deep packet inspection function. The security network service is implemented by forwarding the customer's traffic first through a subscriber gateway VNF instance to establish the subscriber's plan that includes, e.g., enforcing a traffic volume limit. Then, the traffic is forwarded through a firewall VNF instance to block unwanted connections from the Internet and through a deep packet inspection VNF instance to prevent that, e.g., voice over IP (VoIP) is used over the data connection as many mobile operators do today. The first VNF instance is configured to the specific needs of the customer and is, therefore, a dedicated VNF instance that is operated only for a specific customer. Both, the firewall and the deep packet inspection VNF instances, are not specific to one subscriber. The mobile network access service used in the example is created by a network function chain, which defines an order of VNFs the traffic traverses. The specific chain that is implemented in the data center for a specific customer is termed network function chain instance.

The traffic of the customer is identified by one of the ISP's edge switches. It is tagged there and then forwarded through the VNF instances as required by the network service. All services in the edge data center that use NFV rely on the network function chaining application. Therefore, in the edge data center network's control plane, the network function chaining operates at the highest priority. The only other control plane application with the same priority is the fabric application that provides basic connectivity.

Different approaches to implementing network function chaining exist, with different performance and usage characteristics. One central aspect of network function chaining is the identification of network packets after a VNF has processed them. The issue is caused by the fact that in many of today's architectures, a single VNF is designed to service many network function chain instances at the same time. Therefore, an identification mechanism is required to identify packets of a specific network function chain instance after they have been processed and the packets' contents have been potentially modified.

Many existing approaches exist on how to approach this issue. Network service headers introduce a new network protocol that is used by intermediate switches, virtual switches, and VNF instances to keep track on the traffic flows, their affiliation, and their path through the available VNF instances [RFC8300]. However, the design requires the implementation of a new, complex network protocol on every component in the system. The protocol is not available today in all devices, e.g., it is not supported by OpenFlow. StEERING [Zha+13] is a function chaining approach that was specifically introduced and optimized for OpenFlow. However, it does not address the packet identification problem stated above. SIMPLE [Qaz+13] addresses the issue, and while the proposed solution seems promising, it is complex to implement. Therefore, we propose to use network interfaces for traffic identification [Ble+14]. Network functions signal the function chain instance, as well as the direction a packet is forwarded in that chain instance, by sending packets out a specific network interface.

While the approach solves the described problem elegantly, it requires the modification of many flow entries in case of an NFV infrastructure device failure. This is caused by the fact that for every VNF instance, a set of flow entries must be installed on the NFV infrastructure device to which the subscribers of the failed device are moved to. The number of flow entry additions can lead to a performance bottleneck of the flow table memory interface of the virtual switches the subscribers are moved to. The income of ISPs often directly depends to the availability of their network access service. Therefore, moving the network function chain instances for the subscribers affected by the failure to a new NFV infrastructure server is time critical.

We will investigate the performance bottleneck characteristics, and the information required by the network function chaining control plane application. Furthermore, a strategy for mitigating the performance bottleneck is investigated. Specifically, we identify the data center ToR switch that connects the virtual switch of the VNF infrastructure device to the data center fabric as having a low flow entry addition load. Therefore, we investigate the efficiency of applying mitigation strategy ③ to the network function chaining application. We do so by shifting a part of the flow entry addition load from the virtual switch to the ToR switch to decrease the total failover time and thereby increase the application's reliability.

5.2.2 *Software-Defined Multicast*

Network layer multicast, e.g., IP multicast proposed by Deering et al. [DC90] has been investigated in academia and industry for two and half decades. Multicast is successfully used in ISP networks today to provide IPTV services. This service accepts a single TV signal per channel as a packet stream at a data center of the ISP. The network devices duplicate the packets of this stream on their way to the subscribers as needed. By avoiding unnecessary transmissions of the same content, the network traffic is reduced. Unfortunately, IP multicast is used today within single network domains only.

The increasing popularity of OTT content creates a high traffic load on content delivery networks (CDNs) that distribute the content as well as on ISP networks that interconnect CDNs to their customers. Furthermore, the CDN traffic can lead to unexpected traffic patterns at the edge links of ISPs. These can interfere with the traffic engineering process of ISPs and thereby lead to performance bottlenecks in the data path [Ble+18]. Many of these OTT services could benefit from network-level multicast, which is not available on the Internet today. Where IP multicast failed for global content delivery, SDN-based packet replication services have been proposed recently to tackle this issue, such as SDM by Rückert and Blendin et al. [RBH15].

One example of a high traffic load event caused by OTT is the rollout of software upgrades. An investigation of the rollout of updates of Apple iOS devices by Blendin et al. [Ble+18] showed significant traffic increases in the network of a large European ISP within a single day. Furthermore, the observed update event saturated the delivery infrastructure of Apple, which required it to buy additional capacity from third-party SDN providers. From the perspective of ISPs, the event created unpredictable traffic patterns on seemingly unrelated network links. To mitigate these traffic engineering issues for ISPs, SDM could be used to distribute the software update to small caches on the subscriber's premises. Thereby, the load on the delivery infrastructure of Apple could be reduced through network level packet duplication, and the ISPs can increase the control over the traffic in their network. Another area of application that would strongly profit from Internet-wide network layer packet replication and SDN-based network-layer replication support is publish-subscribe systems, as demonstrated by Bhowmik et al. [Bho+17].

The interest in content on the Internet, which is reflected by the distribution of group sizes in multicast systems, is Zipf distributed. This characteristic was discovered by Adamic et al. [AH02] and was confirmed for streaming services by Sripanidkulchai et al. [SMZ04].

Multicasting on the network layer and the layers below is limited by the number of groups a system can control. Figure 5.3 depicts a conceptual view of the Zipf-like distribution of multicast group sizes and which technologies apply to which part of the distribution. A few huge groups such as TV channels can be and are distributed using physical layer multicast, i.e., radio broadcast. While the technical effort to implement

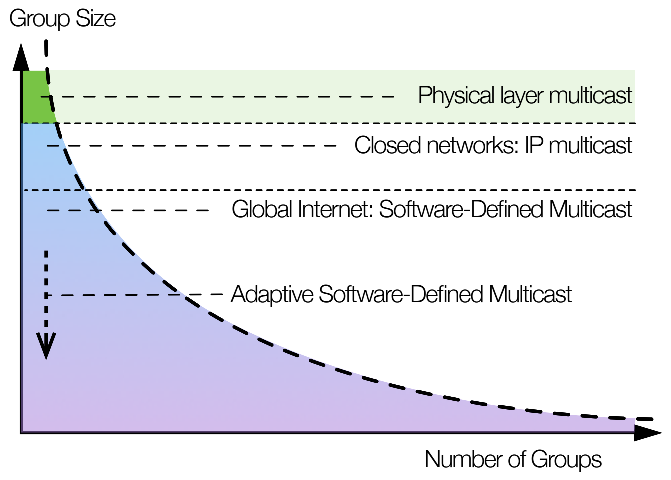

Group Size
Physical layer multicast
Closed networks: IP multicast
Global Internet: Software-Defined Multicast
Adaptive Software-Defined Multicast
Number of Groups

Figure 5.3: Zipf-like multicast group size distribution and multicast delivery technologies

such a system is high and often only tens of channels are supported, its costs are small for a single consumer or multicast group member. The costs are low because an additional consumer or group member in the system does not increase its costs. Smaller multicast groups are often not available for physical layer multicast but could be distributed in closed networks using IP multicast. The latter is done today in IPTV systems of ISPs. In IP multicast per group state is kept in all data plane elements involved in forwarding multicast traffic. This limits the maximum number of groups such a system can host, in addition to the lack of a global multicasting system.

SDM is one approach to for global multicasting on the Internet. It enables global network layer multicast by using unicast addressing and transparently applying multicast forwarding inside of SDM-enabled networks. This, in addition to a well-defined service interface and an end-to-end detection mechanism on the Internet, enables global multicasting without requiring every network on the way to support this mode of forwarding. However, the original SDM approach still relies on the traditional distribution methods that are used by IP multicast as well. This means that the amount of memory required on data plane elements per multicast group and member is high, which limits its application to a small number of large multicast groups. The vast number of small multicast groups observed in Zipf-distributed group sizes cannot be efficiently supported.

ASDM aims to enable just that: supporting a large number of multicast groups, even if they are small. There are two perspectives on this: the general efficiency of the approaches per subscriber in the given Zipf-like group distribution as well as the management of multicast group state in the data plane elements of ISP networks. The general state efficiency of multicasting approach can be optimized in a generic way for all ISPs. However, the sheer number of multicast groups in large-scale multicast systems will always be able to fill the available state memory in the data completely. Therefore, an approach is required that enables ISPs to adapt the behavior of the multicast system to their needs regarding the tradeoff between data plane memory and data rate to be able to utilize them to their full capacity. This includes the requirement for such a system to work with different amounts of available memory in the data plane.

This requirement is strengthened by the fact that SDM is not expected to be a high-priority services in the ISP network control plane. It increases the efficiency of the network and reduces the need for traffic management intervention but is considered an add-on service. From the perspective of resource consumption in the data plane, an approach is needed that is very efficient per group member, supports a large number of groups, reduces the traffic volume in the network, and can operate on the residual data plane memory that is not used by high-priority applications.

To that end, we propose the ASDM and ABSDM approaches that combine the global availability of SDM with a controllable tradeoff between state and data rate utilization in the data plane.

5.3 ENABLING DYNAMIC FUNCTION CHAINING TO MITIGATE FLOW ENTRY ADDITION BOTTLENECKS

The goal of the network function chaining design is to provide an efficient, reliable, and simple approach that can be practically deployed in edge data centers today. To that end, the design introduces an efficient and simple approach for identifying traffic flows coming out of VNF instances, an efficient forwarding scheme, and a scalable VNF instance deployment design. The ability of the design to mitigate control path bottlenecks increases the reliability of the service provided by the application. Before discussing the design of the bottleneck mitigation approach, we introduce the design for the network function chaining approach to discuss the design decisions that affect the design of the mitigation approach.

An example data center that uses the proposed SDN-based the network function chaining design is depicted in Figure 5.4. The NFV infrastructure servers are located in racks and connected to ToR SDN switches. These are in turn interconnected using additional aggregation switches in a leaf/spine topology as proposed for edge data centers by Peterson et al. [Pet+16].

We introduce the design of the proposed system by introducing the flow identification and forwarding scheme in Section 5.3.1. The central packet interface definition of VNF instances is described in Section 5.3.2. Finally, our design for mitigating control path bottlenecks is discussed in Section 5.3.3.

5.3.1 *Packet Flow Identification and Forwarding Scheme*

The core design decision of the approach is the packet flow identification at the packet interface of VNF instances. Network function chain instances use a mix of dedicated

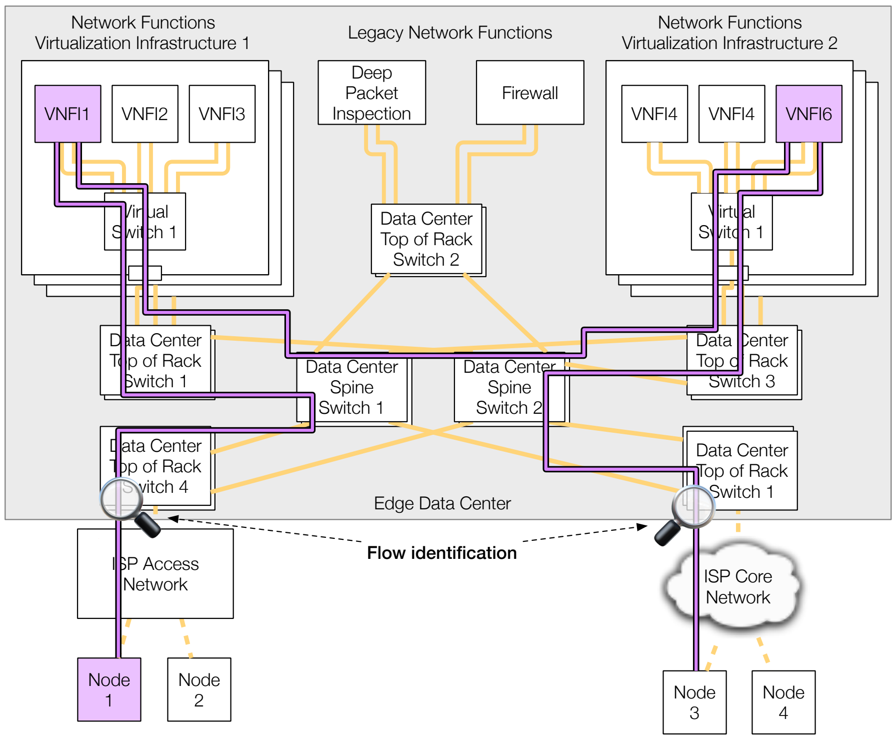

Network Functions Virtualization Infrastructure 1
Legacy Network Functions
Network Functions Virtualization Infrastructure 2
VNFI1
VNFI2
VNFI3
Deep Packet Inspection
Firewall
VNFI4
VNFI4
VNFI6
Virtual Switch 1
Data Center Top of Rack Switch 2
Virtual Switch 1
Data Center Top of Rack Switch 1
Data Center Spine Switch 1
Data Center Spine Switch 2
Data Center Top of Rack Switch 3
Data Center Top of Rack Switch 4
Data Center Top of Rack Switch 1
Edge Data Center
Flow identification
ISP Access Network
ISP Core Network
Node 1
Node 2
Node 3
Node 4

Figure 5.4: An example edge data center with the proposed network function chaining design (adapted from [Ble+15a]).

and shared VNF instances as depicted in Figure 5.4. The system relies on network interfaces to identify traffic sent to and received from VNF instances. In most cases, a dedicated VNF instance has two interfaces: the ingress interfaces for traffic from and to the subscriber and the egress interface for traffic from and to the Internet. This simple approach solves the problem of traffic identification that often arises when a large number of subscribers share one VNF instance [Qaz+13]. The problem is caused by the packet interface characteristics of a type of network functions. Specifically, network functions that modify the packet flow headers in a way that makes them not easily identifiable after processing, e.g., by modifying the IP addresses, or create an entirely new packet flow. An overview of selected packet interface types of VNF is given in Table 5.1, which is discussed in Section 5.3.2. By using network interfaces to identify traffic, the network function chaining problem is reduced to the problem of creating interconnections for network interfaces in the data center. A network function graph instead of a chain can be implemented through VNF instances with more than two interfaces. The additional interfaces enable the instance to select alternative paths in the graph. Shared VNF instances can be implemented by using multiple, separate sets of interfaces per network function chain instance.

Creating many network interfaces is not an issue since the network interfaces are created on software switches and, therefore, are software instances only. The NFV architecture is assumed to be based on many small, disaggregated VNFs. A dedicated VNF instance per subscriber or traffic stream can be used. The idea and feasibility of operating a dedicated VNF instance per traffic flow in the form of a virtual machine (VM) were presented by Bifulco et al. [Bif+13] and confirmed by Madhavapeddy et al. [Mad+15] as well as Manco et al. [Man+17]. The concept is also proposed for the CORD-based design for residential access networks by the ONF[1]. Furthermore, the deployment process for a large number of VNF instances is supported by our design through the concept of VNF instance isolation.

The network function chaining approach is aimed at edge data centers where, in addition to other services, the service gateways are operated for subscribers. These service gateways could be implemented as virtual machines as proposed by the ONF in their R-CORD proposal for implementing residential network access services[2] or based on hardware switches as proposed by Nobach and Blendin et al. [Nob+17]. Independent of the implementation, subscriber identification facilities are available, either as IP addresses or prefixes or in the form of virtual LAN (VLAN) or other protocol tags. These identification facilities are used for subscriber identification and can be used for matching using the OpenFlow protocol [Nob+17]. In this discussion, we use the subscribers IP addresses. However, the mechanism can be replaced without changing the characteristics of the design. Traffic coming from the Internet of other parts of the ISP network is always identified by its IP address.

The forwarding scheme is inspired by the MAC-address based data center forwarding approach PortLand proposed by Mysore et al. [Nir+09]. We use MAC addresses instead of VLAN tags [IMS13] or Segment Routing because it is well supported by existing OpenFlow hardware. This includes masked matching of MAC addresses as well as writing MAC addresses. Support for, e.g., VLAN tags in existing SDN hardware is not complete. As shown, e.g., by Nobach et al. [Nob+17], even state-of-the-art switching ASICs such as the Trident II do not fully support the processing of more than one VLAN tag. While the PortLand approach fits the edge data center use case well, it is not designed for SDN and is a generic design for data centers. Since edge data centers are restricted in size and we can simplify the forwarding mechanism.

Our proposed MAC address encoding scheme is depicted in Figure 5.5. A modified

$$\underbrace{00}_{\text{Rack ID}}:\underbrace{09}_{\text{VNFI ID}}:\underbrace{2d:01}_{\text{VNFI Port ID}}:\underbrace{03:02}_{\text{Flow ID}}$$

Figure 5.5: The MAC address encoding of forwarding information (adapted from [Ble+15a]).

1 https://wiki.opencord.org/pages/viewpage.action?pageId=1278090
2 https://www.opennetworking.org/r-cord/

version of the hierarchical addressing scheme of PortLand is used. The first byte of the 48-bit MAC address field encodes the rack in the data center where the NFV infrastructure server is located. The second byte signifies the server in the rack that is addressed. The 12 bits after that specify the port number of the virtual switch of this server. The last 20 bits encode a unique flow ID that can be used for verification by VNFs and debugging purposes; it is not required for the forwarding process.

The maximum edge data center size supported by this design is 256 racks with 256 servers each for a total of 65,536 servers. Each NFV infrastructure server can host at maximum 32,768 VNF instances with two interfaces each. We argue that these restrictions are sufficient for edge data centers.

This design combines the advantages of hierarchical addressing with the freedom for forwarding customizations for the leaf-spine layer. We assume a fully redundant leaf-spine architecture, where every VNF instance is connected to two ToR switches that are in turn connected to all available spine switches. Specifically, the ToR switches that act as leaf switches and the spine switches need at most 512 forwarding table entries for redundant paths to any NFV infrastructure server. For directly connected servers, ToR switches additionally require 256 forwarding table entries. Because at each hop, only a part of the MAC address is matched against, wildcard matching is required. State-of-the-art OpenFlow switches support TCAM-based tables of these sizes.

In the investigated NFV environment, most interfaces are virtual interfaces. The virtualization manager assigns the MAC addresses of virtual interfaces. Hence, in this specific use case, we can control all MAC addresses of the involved devices and assign them as required. Therefore, optimizations or more general approaches such as the „MAC address as routing label“ approach proposed by Schwabe et al. [SK14] are not required.

5.3.2 *Virtual Network Function Instance Packet Interface*

A packet interface defines the data plane interface between the network function chaining system and attached VNF instances. This interface is crucial to the presented system because it implements the network function chaining as well as the VNF instance isolation feature.

The chaining of network functions relies on matching on the ID of the port a packet is received on from a VNF instance. This port ID is sufficient to determine the next VNF instance the traffic has must be forwarded to. Therefore, the virtual switch has to match for the incoming port ID and write the encoded identification of the port of the next VNF instance in the chain into the MAC address field of the packet. Packets that are sent to VNF instances only need to be matched for the part of the destination MAC address that encodes the destination port ID.

The VNF instance isolation feature is depicted in Figure 5.6 simplifies the deployment of large numbers of VNF instances. Normally, depending on the type of VNF, each

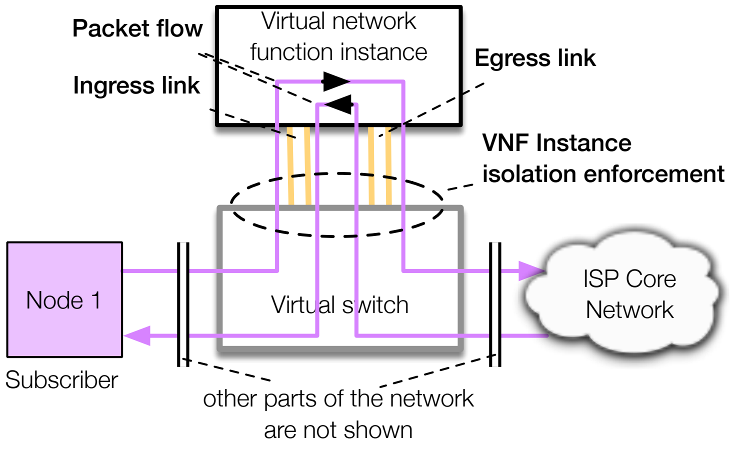

Packet flow
Virtual network function instance
Egress link
Ingress link
VNF Instance isolation enforcement
Node 1
Subscriber
Virtual switch
ISP Core Network
other parts of the network are not shown

Figure 5.6: The network function chaining system's packet interface to VNF instances (adapted from [Ble+14]).

instance needs a unique MAC address, IP address, as well as knowledge about the next hops for forwarding traffic. To that end, we propose to completely isolate the VNF instances from network control traffic, by filtering and manipulating Address Resolution Protocol (ARP) messages. Thereby, all VNF instances can use the same network address information if needed. Alternatively, the MAC addresses for each VNF instance can be configured to be the address used by the proposed forwarding system.

Implementing the isolation feature requires additional packet processing on the virtual switch. First, all control traffic is filtered, or rate limited and forwarded to a centralized MAC address manager. This manager provides MAC addresses for next hops the VNF instances might want to forward traffic to. When packets are sent to or received from isolated VNF instances, the exact processing depends on the type of VNF. Therefore, we derived a classification for VNF packet interfaces and their specific packet processing requirements to providing instance isolation that is listed in Table 5.1. The column „Forwarding" describes the type of packet processing conducted by the VNF instances. The column „In: requirements" specifies the required MAC address configuration for incoming packets to be forwarded. The modifications applied by the VNFs are listed in the column „Modified fields". The modifications applied by the network function chaining system to packets received from the VNF instances and to be forwarded towards the internet are shown in „Internet→Subscriber: packet modifications". If a VNF type can be shared between chain instances without losing its identifying information, the column „Sharing possible" contains the entry „yes".

Table 5.1: Typical network function types and their packet interface types [Ble+14].

Name	**Forwarding**	**In: requirements**	**Modified fields**	**Internet→Subscriber: packet modifications**	**Sharing possible**	**Example VNF**
Transparent	port based	none	none	none	yes	intrusion detection system
L2	bridging	$MAC_{dst} \neq MAC_{if}$	none	none	yes	switch
L3	routing	$MAC_{dst} = MAC_{if}$	L2 src,dst	$L2_{dst} = MAC_{next\ instance}$	yes	firewall
$L4_{dst}$	routing	$MAC_{dst} = MAC_{if}$	L2 src,dst; L3,L4 dst	L3 modifications	yes	destination NAT
$L4_{src}$	routing	$MAC_{dst} = MAC_{if}$	L2 src,dst; L3,L4 src	$L3_{src} = IP_{user}$ + L3 modifications	no	source NAT
L7	proxying	$MAC_{dst} = MAC_{if}$	L2,L3,L4 src,dst	$L4_{src}$ modifications	no	HTTP proxy

5.3.3 *NFV Infrastructure Failover Bottleneck Mitigation*

Investigating potential control path bottlenecks for the proposed network function chaining system leads to the virtual switches as the primary location of memory consumption and control plane activity. The only other component in the system where considerable flow table space is used is the ToR switches that hand over traffic to subscribers or the core network. The distribution of required OpenFlow flow table space and the number of flow entry modification required for modifying a network function chain instance is listed in Table 5.2. The core switches only require modifications in the case when core switches or ToR switches are removed or added.

Table 5.2: OpenFlow flow entry metrics (adapted from [Ble+15b])

Metric	**Switch type**	**Number of flow entries**
Max. number of entries per switch	Edge switch	2 (Num. subscribers in the system) (IP Addresses per subscriber)
	ToR/core switch	2 (Num. ToR switches in use)
	Virtual switch	$4\,\Sigma_{i\in\{\text{Local VNFI}\}} \text{VNFI}^i$ (Num. subscribers)
Max. entry modifications per subscriber modification	ToR switch connected to the ISP core or access network	2 (Num. IP addresses of subscriber)
	ToR/core switch	None
	Virtual switch	4 (Num. of local network function instances of subscriber)

Considering the resilience of the leaf-spine configuration the only event that requires a large number of flow entries to be modified is an NFV infrastructure server failure. In this case, all VNF instances that are hosted on this server must be relocated to a different server. The total service outage time depends on the time required to move the VNF instances as well as the time to redirect the traffic flows by modifying the network function chain instances.

When the server has already crashed, migrating the VNF instances is not an option. Instead, they must be re-created. The creation time for per-chain instance VNF instances can be as low as tens of milliseconds as reported by Manco et al. [Man+17]. Unfortunately, it is not clear from the paper of Manco et al. how well this process can be parallelized. Their investigation show constant per VM creation times below 10ms. However, it is not clear if using the 60 CPU cores like in their experiment to create 3000 VMs result in a total sequential time of $10ms * 3000 = 30s$ or parallelized $\frac{30s}{60cores} = 0.5s$. The parallelized completion time of 0.5s is considered an acceptable service outage duration, 30s is not. If this latter case is true, the failover time for the VNF instances can be reduced by keeping enough VMs of every type available that a single server can fail. In the failure event, the existing VNF instances only need to be configured for their network function chain instance. We expect this time to be much lower than the time needed to create a new VM.

In either case, new OpenFlow entries on the virtual switches must be created in order to repair the broken function chain instances. As listed in Table 5.2 for every moved VNF instance four flow entries must be added. This creates a spike in control path activity on the virtual switch, in case of 3000 VNF instances 12,000 flow entries must be added.

Depending on the performance of the flow addition process on the virtual switch, this can take a considerable amount of time. Since we assume that the network function chaining application operates at the highest priority in the control plane, interference from other applications has not to be accounted for. The control paths of the two ToR switches every virtual switch is connected to are entirely idle. Therefore, the system is designed so that it can share the flow entry addition load between the virtual switch on the NFV infrastructure server and its adjacent ToR switches.

The idea is that some of the flow entries can be installed on the hardware switch as well, and then are later moved to the virtual switch once the control path load is back to normal. The process ensures that the data path of the packet is not modified to conduct control path load sharing. This restriction requires that only flow entries are moved to the ToR switch that would pass through it anyway. For example, flow entries that forward packets between VNF instances inside of the virtual switch are not eligible to be installed on the ToR switches. Therefore, only chain links between VNF instances on different servers to chain links to the handover ToR switches to the access and core network are eligible for moving. A flow entry must be added to the virtual switch that forwards all packets, which are received from local VNF instance ports and are not handled by local flow entries, to one of the connected ToR switches. To that end, a flow entry is added that writes the correct source MAC address in every outgoing packet to ensure that the information on the switch port the packet was received on is available for matching on the ToR switch as well. This behavior can be implemented by copying the source port ID to the corresponding MAC address field using the *Copy-Field* primitive of OpenFlow. Since the virtual switch can forward the packets to any of the two ToR switches or might even use equal-cost multi-path (ECMP) routing, the flow entries must be added to both switches.

The preparation for the mitigation for control path resource bottlenecks that affect the flow entry addition performance starts when the NFV orchestrator initializes the network function chaining application. The application gets all relevant information from the controller, e.g., the network topology, and the link speeds. The virtual switches, as well as the ToR switches that are connected to the ISP access and core networks, are identified, and configured. Then, the application queries the controller for the control path characteristics of the virtual switches as well as the ToR switches adjacent them. The controller provides the answer including a measure for the uncertainty in the flow entry addition performance. The control plane application estimates if, in the best case, sharing the flow entry addition load can decrease the completion time of the process sufficiently to trade off the overhead of the process. Since the performance of TCAM-based flow tables depends on many factors, e.g., the number of existing flow entries in the table, this estimation is only used to decide of load sharing makes sense at all. If load sharing can improve the completion time, a process is activated that decides when the actual event occurs if it still makes sense given the specific information. If load sharing cannot improve completion times at all, the decision process in case of a server failure is disabled.

When the NFV orchestrator signals that a server has failed and that a set of flow entries must be installed on an NFV infrastructure server, it also provides a deadline for the completion of the process. The application immediately requests the current flow entry addition performance estimated from the controller. It then analyses, which and how many of the flows are eligible for being moved to the ToR switch. Now, the decision is made if load sharing reduces the completion time sufficiently. If this is the case, the application first installs the default route for forwarding unmatched packets from the virtual to the ToR switch. Then the bulk of the flow entries are sent to the controller with a request to receive constant updates on the installation processes performance.

If the flow entry addition performance on either of the switches does not match the values provided by the controller, the application might must re-run the decision process and, e.g., modify the load sharing ratio. Once the application received the confirmation that all operations are completed, it forwards this information to the NFV orchestrator and the process is finished.

Literature suggests that flow entry additions on software switches are much faster than on hardware switches using TCAM-based flow tables [HYS]. The application needs a mechanism to decide whether moving a part of the flow entries to the ToR switches is worth the overhead.

We express the completion time improvement for adding all flow entries in Equation (5.1).

$$R = \frac{\max\left(\frac{M}{\mathrm{tp}_{\mathrm{ToR}}(M)} + O, \frac{N-M}{\mathrm{tp}_{\mathrm{vS}}(N-M)}\right)}{\frac{N}{\mathrm{tp}_{\mathrm{vS}}(N)}} \tag{5.1}$$

where:

$R \in [0,1]$	= relative reduction in completion time
N	= Total number of flow entry addition operations
O	= Processing time overhead
E_{vS}	= Data plane element: virtual switch
E_{ToR}	= Data plane element: ToR switch
$\mathrm{tp}_{E_i}(n)$	= Average throughput for n flow entry addition operations
M	= The number of flows that are moved to the ToR switch
$M_{\max}$	= The maximum number of eligible flows for moving
$M_{\max} < N$	= Number of flow addition operations moved to E_{ToR}

Literature shows that the throughput of adding flow entries to TCAM-based flow tables decreases with a growing number of entries in the table. Therefore, the flow entry addition throughput is not given as a fixed number but as a function.

The number of flow entries M to be moved to the ToR switch is derived as provided in Equation (5.2).

$$\min\ D(M) = \mathrm{abs}\left(\frac{N-M}{\mathrm{tp}_{\mathrm{vS}}(N-M)} - \frac{M}{\mathrm{tp}_{\mathrm{ToR}}(M)} - O\right) \tag{5.2}$$

subject to:

$$0 \leq M < N$$
$$M \leq M_{\max}$$

5.4 MITIGATING FLOW TABLE SPACE SHORTAGES WITH ADAPTIVE SOFTWARE-DEFINED MULTICAST

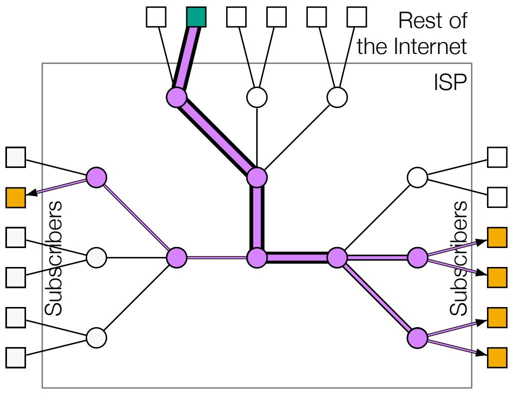
The design of ASDM and ABSDM aim to provide a multicast system that fulfills the requirements for control plane applications in ISP networks: efficiency, reliability, and simplicity. Simplicity is achieved by carefully designing the system to use one single configurable parameter. Reliability is provided by its ability to optimize the system either for minimizing state on the data plane or minimizing the data rate of the forwarded traffic. ASDM and ABSDM are efficient because their adaptive design enables the optimization of the OpenFlow flow entry consumption. Thereby, the approaches can operate as add-on control plane applications with lower priority in the ISP network's control plane. Furthermore, while ASDM is designed to operate on a unicast forwarding underlay, we show by applying the adaptive multicast concept to the highly efficient bit-indexed replication forwarding underlay in our ABSDM design, that the concept can be applied to other forwarding concepts as well.

The general approach to control the tradeoff between state memory and data rate usage is presented in Section 5.4.1. The routing design and the corresponding approach to constructing the multicast tree of ASDM on unicast underlay networks is described in Section 5.4.2 and the approach of ABSDM for bit-index replication underlays in Section 5.4.3. Finally, the interactions and requirements of both control plane applications with the controller to mitigate global memory shortages in SDN networks are discussed in Section 5.4.4.

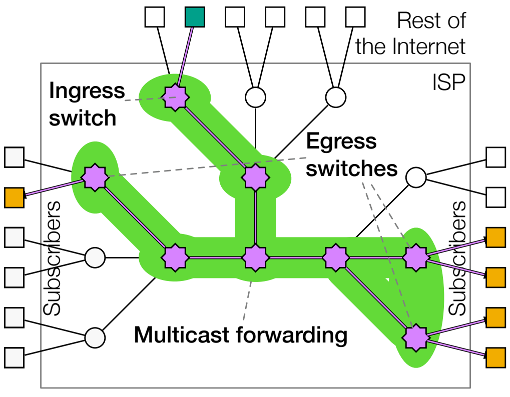
5.4.1 *Resource Adaptation Approach*

The goal of the ASDM and the ABSDM systems is to provide an SDM-compatible forwarding mechanism that can host a large number of multicast groups where the group sizes are Zipf-distributed. We will discuss the general approach to adaptive multicasting in this section on the example of ASDM. We build on the SDM approach in that the external interface of the system is the same. This means that outside of the network domain the SDM control plane application controls, all traffic is unicast traffic. However, within the system, multicast forwarding is applied to the traffic, which includes network layer packet replication. Multicast groups are managed through a service interface that operates in the service and controls the control plane application. This management interface is part of the SDM approach. ASDM replaces the multicast forwarding inside of the controller network domain, all the other parts of the SDM system stay the same. We discuss ASDM on the background of SDM. Traditional IP multicast is not considered a viable alternative. Rückert et al. provide a detailed comparison between IP multicast and SDM in [RBH15]. For even more detailed discussion on the subject and an exhaustive presentation of the related work on the issue of multicasting, refer to the Ph.D. thesis of Rückert [Rüc16].

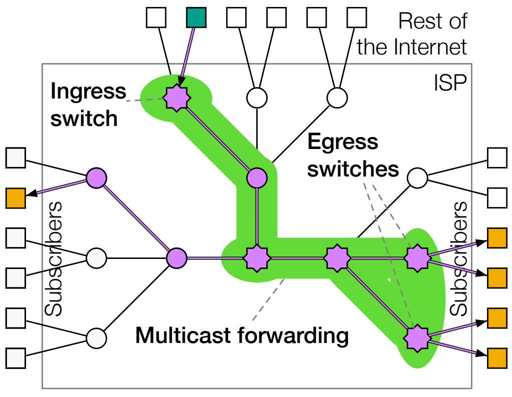
Building on the SDM concept, ASDM relies on unicast addressing outside of the controlled network. However, in contrast to SDM, unicast is not only used on the edges of the network, but also inside of the ISP network as depicted in Figure 5.7c. For a single multicast group, as with SDM, packets are addressed by the traffic source, denoted by 🟩, to a specific data plane element near the edge of the ISP network, termed ingress switch. This data plane element is responsible for identifying packets as SDM traffic and its group affiliation and tagging the packets accordingly. From then on, the forwarding differs between SDM and ASDM. While SDM employs a special multicast forwarding scheme, ASDM relies on unicast forwarding between multicast-enabled data plane elements. The first node matches the incoming packet, determines the next multicast switches, duplicates the packet if required and addresses each copy of to the next multicast switch on its respective path. On these switches the process is repeated until the edge of the multicast forwarding system is reached. Then, a similar process is applied, but this time using the destination addresses of the multicast group members, denoted by 🟧, as specified by SDM. Using unicast between the multicast switches within the ISP network has the advantage that only selected switches must keep state for this specific multicast group. In contrast to these switches that host multicast state, denoted by ✸, other data plane elements on the packets' forwarding path are not involved with multicasting, denoted by ●, and only host unicast forwarding state. Traditional multicast forwarding methods, such as IP multicast and SDM require per-group multicast state on every switch along the forwarding path between the source, and the group members, as depicted by the area in Figure 5.7b highlighted in green. Stoica et al. first proposed this approach in their recursive unicast approach to multicast (REUNITE) paper [SNZ00].



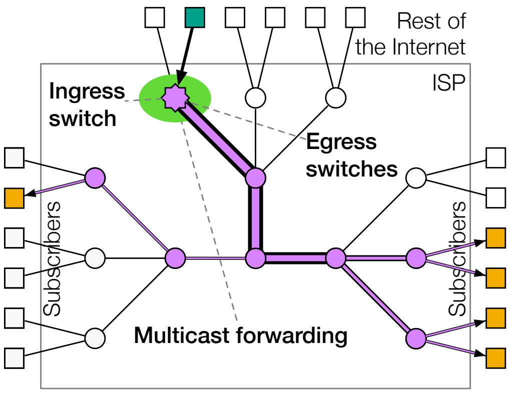
(a) Unicast



(b) Default SDM multicast



(c) ASDM late replication



(d) ASDM early replication

Figure 5.7: Packet replication mechanisms in comparison to unicast (adapted from [Ble+15b]).

ASDM builds on this concept to make the tradeoff between data plane memory and data rate in multicasting, which was first investigated by Radoslavov et al. [REG99], controllable for the control plane application. It does so by exploiting the logically centralized control plane to adapt the multicast forwarding to the resource availability in the data plane it operates on. As described before, Figure 5.7c depicts the traditional multicast approach. The replication happens at the last possible location in the multicast tree, thus termed late replication. Late replication is the traditional multicast strategy that solely aimed at reducing the total transmission volume of the system. The data rate is the same on all links, denoted by the smallest of the filled lines ▬. However, there are four nodes in the system that host multicast state and only three nodes that forward the multicast traffic without hosting the corresponding group state. The minimal subgraph that contains all multicast switches of a group, highlighted green in the depiction, is termed to do multicasting, the other parts of the system do unicasting. The area where multicasting is used is said to convert outgoing packets to unicast at their edges, the switch that performs the unicast conversion is termed egress switch.

In contrast to late replication, early replication immediately converts packets to unicast traffic on the first node on the ISP network as depicted in Figure 5.7d. The consequence of this forwarding strategy is that the data rate is higher on many of the links. However, there is only one node that hosts state for this multicast group and six nodes that perform unicast forwarding only. Consequently, the multicasting area for this group is made up of a single data plane element only. The system can implement every configuration between these two strategies, early and late duplication. This range of configuration options gives the system its freedom to select the tradeoff between a high data rate and low multicast state with early replication and a low data rate with a high multicast state with late replication. Despite the freedom the system provides regarding resource consumption, it requires only a single parameter to regulate its behavior: the unicast conversion threshold.

Nevertheless, in both cases, the crucial handover point between the ISP and its neighbor is relieved from high data rates. Hence, independent of the selected replication scheme, ASDM implements a packet replication service, while its forwarding characteristics inside of the controller network can be close to unicast traffic. In contrast to that, unicast requires the full traffic volume on this link as well, as depicted in Figure 5.7a.

5.4.2 *Adaptive Multicast Routing*

The unicast conversion threshold T is the only parameter to configure the ASDM control plane application. It controls the tradeoff between the consumption of matching memory and data rate not only of a single group but for the complete system. The goal behind this design decision is to provide an effective way to control the system which is simple enough to be understood by its operators when used in the context of an ISP network with tens or hundreds of other control plane applications. The unicast conversion threshold denotes that if the number of members in a subtree is equal or less, the traffic for this subtree will be converted to unicast instead of continuing to forward it as multicast traffic.

We explain the unicast conversion threshold and its usage by example in the following paragraphs. ASDM receives the group definition consisting of the traffic source of the group and the group members from the SDM management service. Based on this information, the ASDM application constructs a delivery tree for its members and derives the actual multicast tree that is installed in the data plane from that. The delivery tree consists of all data plane elements involved in the forwarding of the traffic starting from the ingress node as depicted in Figure 5.8. ASDM creates this directed tree by determining the shortest-path tree that originates from the ingress switch to all group members. Since the group members are expected to be either a subscriber of the ISP or located in other parts of the Internet, the tree is created with the last switches in the ASDM network on the paths to the group members as its leaves. The group members are identified by their IP addresses in the SDM system. On each data plane element starting from the ingress switch, each outgoing link is tagged with the outgoing port number of the switch and the group members that are connected through this link as depicted in Figure 5.8.

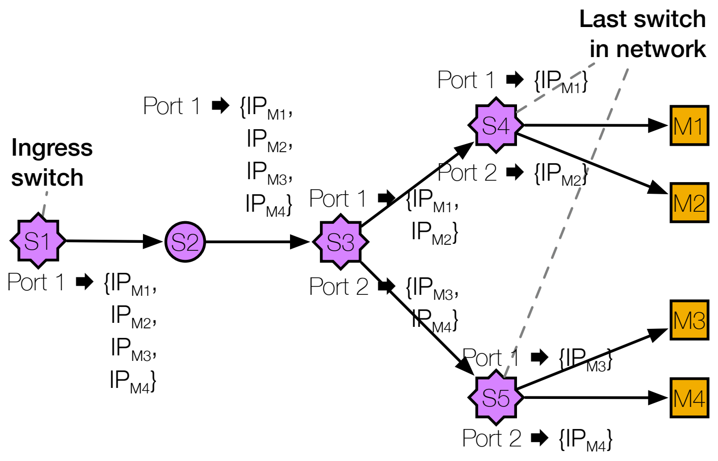


Figure 5.8: The annotated delivery tree used for constructing the ASDM multicast tree with $T = 1$ (adapted from [Ble+15b]).

Then, the system applies the unicast conversion threshold and other constraints if required. The unicast conversion threshold is provided by the system operator or a management system. Its application starts with the ingress switch and checks for every successive link tag if the unicast conversion limit has been reached or a last switch in the controlled network has been found. The unicast conversion threshold for the example system configuration depicted in Figure 5.8 is $T = 1$. Switch $S1$ is the ingress switch and hosts the multicast state for the investigated group. The number of group members in the subtree of its only outgoing port 1 is four, which is higher that $T = 1$. Therefore, only a single action is installed that forwards traffic to the next multicast switch. Which switch that is, is not known at this point in time. When investigating switch $S2$ the system finds that no multicast state is required and moves on. The next investigated switch in the tree is $S3$. Here, two outgoing ports are found, with both subtrees having a higher group member number than T. However, since there are two outgoing links, this node must host multicast state and install an SDN rule that matches incoming traffic for this group, duplicates the group's packets and forwards them to the next multicast switch. At the same time switch $S3$ is the next multicast switch for switch $S1$ and the information for the flow entries there are updated to address all forwarded packets to $S3$. On switch $S4$ for each outgoing link, both the number of group members in the remaining subtree is 1, and the switch is the last data plane element in the controlled network. Therefore, SDN rules are installed on switch $S4$ to convert the traffic to unicast. The same happens on switch $S4$. If a higher value for the unicast conversion threshold is used, the process terminates earlier, and fewer switches are included in the multicasting process. A unicast conversion threshold of ∞ converts all multicast traffic immediately to unicast traffic independent of the group size.

When a group member is added, the topology changes or the value of the unicast conversion threshold is changed, the same process is run again. The result is compared with the existing multicast tree as it exists in the data plane. Then a step-by-step process is derived to move from the existing tree to the new tree. Finally, the new tree is written into the data plane before it is activated.

The advantages of using the unicast conversion threshold over alternatives, e.g., limiting the depths of multicast trees, are manifold. First, a branching limit, where the depths of the multicast trees is limited, is topology dependent. Therefore, it would have to be modified for every single network topology with different shortest-path tree depths it is used in. In contrast to that, the unicast conversion threshold is independent of the network topology. Second, the tree depth limit is also group-size dependent, small groups have a smaller depth and therefore need a smaller limit. Again, the unicast conversion threshold works independently of the group size. Instead, it converts small groups immediately into unicast traffic and is thereby the most efficient when applied to small groups, while its impact on huge groups is much smaller. The unicast conversion threshold, therefore, has precisely the properties it should have to reach our goal of being able to host a large number of small multicast groups. The described tree construction process can easily be extended by additional constraints. For example, intermediate switches in the tree that should not or cannot host multicast state can be easily skipped by the system.

ASDM relies on IPv6 to address multicast switches as well as encode multicast group IDs. REUNITE [SNZ00] proposes to use IPv4 addresses for identifying the switch and User Datagram Protocol (UDP) ports to identify multicast groups. While this approach was sensible at the time when it was conceived, it cannot be considered anymore. Besides limiting the number of multicast groups hosted in a single device to 65,535 sending Internet Control Message Protocol (ICMP) message, used to signal errors, cannot address UDP ports. IPv6 addresses are 128 bits long and can provide, e.g., 2^{64} multicast groups per switch, if a /64 suffix is used to identify groups. While the ratio of IPv6 traffic is still small, many services and clients can communicate via IPv6. Furthermore, the adoption of IPv6 continues to grow, e.g., as reported by Pujol et al. [PRF17], and is expected to be prevalent by the time ISPs adopt SDN.

Each potential multicast switch in the network uses at least one /64 IPv6 prefix for ASDM addressing, other prefixes can be used but are not discussed. Either this prefix is part of an IPv6 subnet assigned to an existing interface, or it is assigned to the loopback interface. The latter is a common approach used by routing protocols, e.g., Open Shortest Path First (OSPF) and Segment Routing [Fil+18]. For efficiency, the system should rely on addresses for which routes that are already available in the network. The second part of the address is the multicast group ID, which is assumed to be the matching /64 suffix of the /64 IPv6 prefix used for addressing as listed in Table 5.3. Thereby, the addresses used by ASDM uniquely identify each multicast group on any given data plane element. The advantage of this addressing scheme is that its usage for routing protocols well understood. Furthermore, is it compatible with the *locator:function* used by Segment Routing that is expected to be used in future ISP core networks [Ble+16b]. The given prefix lengths are examples and can be adapted to the operator's need without affecting the system.

Table 5.3: Example IPv6 address allocation for ASDM subnets (adapted from [Ble+15b]).

Network prefix:	2001:db8::/32
Prefix for switch 0xabcd:	2001:db8:abcd::/48
ASDM subnet:	2001:db8:abcd:8000::/64
ASDM address for group 0x1234:	2001:db8:abcd:8000::1234

As discussed, ASDM is designed to operate on any forwarding system that relies on unicast forwarding, such as OSPF and Segment Routing. In general, the ASDM concept can be transferred to many forwarding substrates. However, the system is restricted in its freedom for selecting forwarding devices by the underlying forwarding scheme.

Bit-indexed replication as proposed by the BIER approach [RFC8279] provides a method for forwarding multicast traffic of many groups without the need to keep per group state on any intermediate switch. Instead, per-group state is required only at the ingress and egress switches. ASDM can be easily adapted to operate on the bit-indexed replication forwarding method instead of unicast forwarding while keeping many of its favorable characteristics. We provide a design for adding adaptiveness to the bit-indexed replication forwarding concept applied to SDM called ABSDM in Section 5.4.3.

More details on ASDM are described in [Ble+15b]. An in depths description of the system, as well as a proposal for a service API, can be found in the thesis of Volk [Vol14].

5.4.3 *Adaptive Bit-Indexed Software-Defined Multicast*

BIER [RFC8279] is a multicast forwarding scheme that uses bit IDs to address multicast destinations. BIER builds on the Xcast idea by Boivie et al. [RFC5058] and the Xplicit SDM idea by Blendin [Ble13] and extends them to a complete multicast forwarding protocol. We, in turn, build on the BIER concept, apply it to SDM and add the adaptive multicast feature to it.

As with the other SDM designs, packets are identified at the edge of the network. If the packet is addressed to a specific group, it is assigned a bit index forwarding header with a bit field that includes a bit for every potential destination in the system. A destination receives the packet if its corresponding bit is set to 1 in the bit field. The bit-indexed addressing scheme works only within the specific network domain, similar to other local-addressing approaches such as MPLS or Segment Routing.

To avoid the issue of Xcast, which only allows small multicast groups because of the per-packet overhead of the multicast destination encoding, we adopt the approach to split a bit string into smaller segments proposed by BIER [RFC8279]. The approach is depicted by example in Figure 5.9. For example, a bit string with a length of 9 bits represents all addressable devices in a network domain. This bit string is considered too big to be included in the bit field header of each packet. Therefore, it is split into smaller bit fields with a maximum length of 5 bits, termed bit sets, to limit the per packet addressing overhead. To that end, the 9-bit long bit string is split into three-bit sets with a length of three bits each. In addition to the three bits to address the multicast switches, a two-bit set ID is used to encode an offset to their respective part of the bit string. In the example, the bit string addresses switch 5, 6, and 9. Switches 5 and 6 are part of the same bit set, which is identified by its offset, 1. Switch 0 is part of bit set 2. To determine the bit indexes the address of Packet 1 encodes, the bit set ID is multiplied by the bit set length and added to the indexes of the individual bits in the bit field. ABSDM is

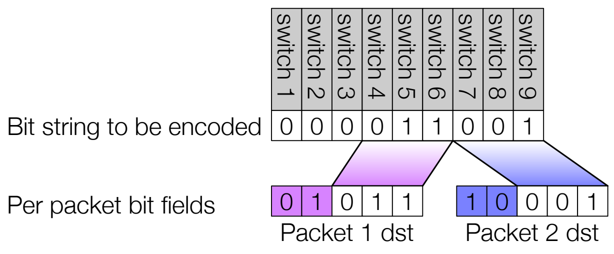


Figure 5.9: Encoding of the global bitfield in per header fields

configured with a global bit string length and the per-packet bit field length. The size of the bit sets SL and the corresponding set ID or offset length OL can then be derived by the system through the fact that the length of the multicast address field AL in each packet is the sum for the set length and the offset length: AL = SL + OL. Specifically, for each bit string length BL the optimal set length is calculated as follows:

$$\min(\mathrm{SL})$$

subject to:

$$2^{\mathrm{OL}} \geq \frac{\mathrm{BL}}{\mathrm{SL}}$$
$$\mathrm{AL} = \mathrm{OL} + \mathrm{SL}$$

An example of an ABSDM forwarding system is provided in Figure 5.10. Only switches that should receive multicast traffic are assigned bit IDs to minimize the total length of the bit string. The multicast forwarding includes all data plane devices in the system and requires multicast state on every one of them. However, this resource consumption is offset by the fact the per-group state is smaller than in other systems because all groups share the forwarding tree. The forwarding tree is created in the same way than described for ASDM with using late replication.

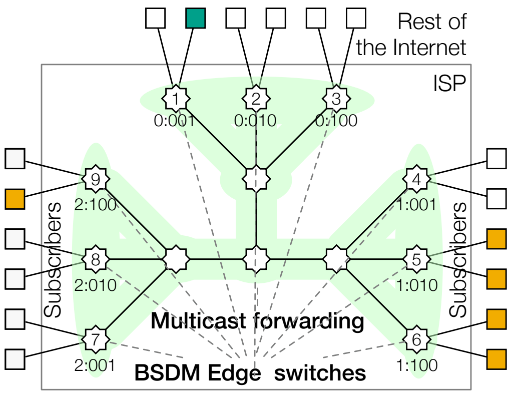
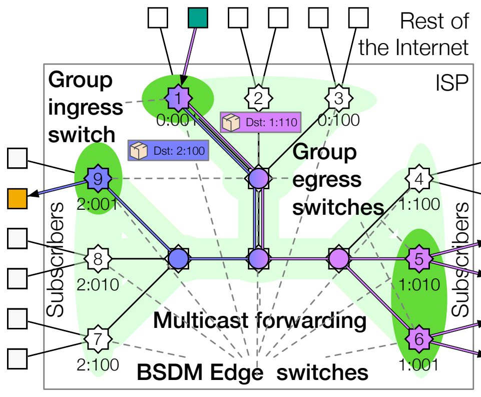
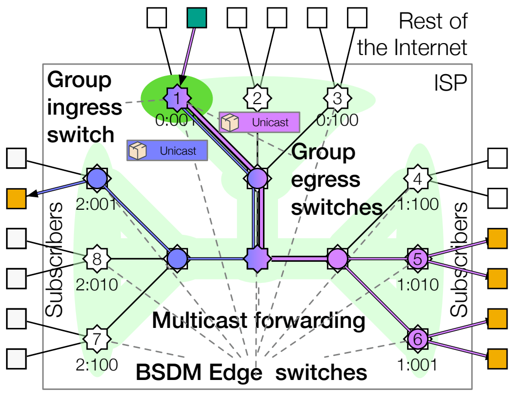


Figure 5.10: BSDM forwarding underlay

An example of a multicast group is given in Figure 5.11. The light green shaded areas depicted the shared forwarding state. The default forwarding method proposed by BIER is late replication that is depicted in Figure 5.11a. As with ASDM, the packets addressed to the group are identified at the group ingress switch. The group is the same used in the bit address encoding example. The destination addresses of the multicast group are part of two different bit sets, 1 and 2. Therefore, the packet is replicated on the ingress switch, a multicast address header is added to each of them, and each copy gets a different multicast address. One packet is addressed to switches 5 and 6, which are part of bit set 1 and one is addressed to switch 9, which is part of bit set 2. The multicast forwarding routers contain forwarding rules for all bit sets, which forward the packets in parallel for the first to two hops. Once the packets arrive at their destination switch, a rule on each of the switches identifies the multicast group, replicates the packets if required, removes the multicast address header, and rewrites the destination addresses to their respective group's members, as with ASDM. As is visible in Figure 5.11a, the number of switches that host per-group state is significantly reduced.



(a) BSDM late replication



(b) ABSDM replication with $T = 2$

Figure 5.11: ABSDM forwarding.

The per-group state can be further reduced by applying adaptiveness to the multicast forwarding as depicted in Figure 5.11b. The unicast conversion threshold in the example is $T = 2$, which means the multicast subtrees of the corresponding bit set are converted to unicast when the number of members they contain is less or equal than 2. Therefore, the traffic for both bit sets is immediately converted to unicast, which leads to a single rule on a single switch only.

The forwarding pipeline of ASDM is more complex than the one for ASDM, which is why we describe in detail in Figure 5.12. It requires an OpenFlow flow table for every port and is assumed to start in the flow table pipeline at flow table m. The depiction shows the process for n ports where ABSDM downstream devices are connected. ABSDM traffic is identified at the beginning of the pipeline and directed to flow table m. This first table is responsible for terminating the multicast process early to implement adaptiveness. We will discuss this table later.

The default bit-indexed forwarding works as follows for every port. For each switch that is reached through a specific port, its bit index is checked in the respective bit set if it is set to 1. In this is the case, the packet is forwarded out this port. To that end, the packet is sent to both, an indirect group table and the flow table of the next port, i.e., flow table m+1. In the group table all bit fields in the set at set to 0 that are not reached through this port. This prevents that the packet is replicated to these addresses again on the next multicast switch, which would cause duplicate packets. Then the packet is output to the corresponding port. The actions applied to the packet in the group table does not affect the copy of the packet that has been forwarded to table m+1. In flow table m+1 the same process is applied, which ensures that the packet is checked if it has to be sent out of every port on the switch. Afterward, the packet is dropped. Note that

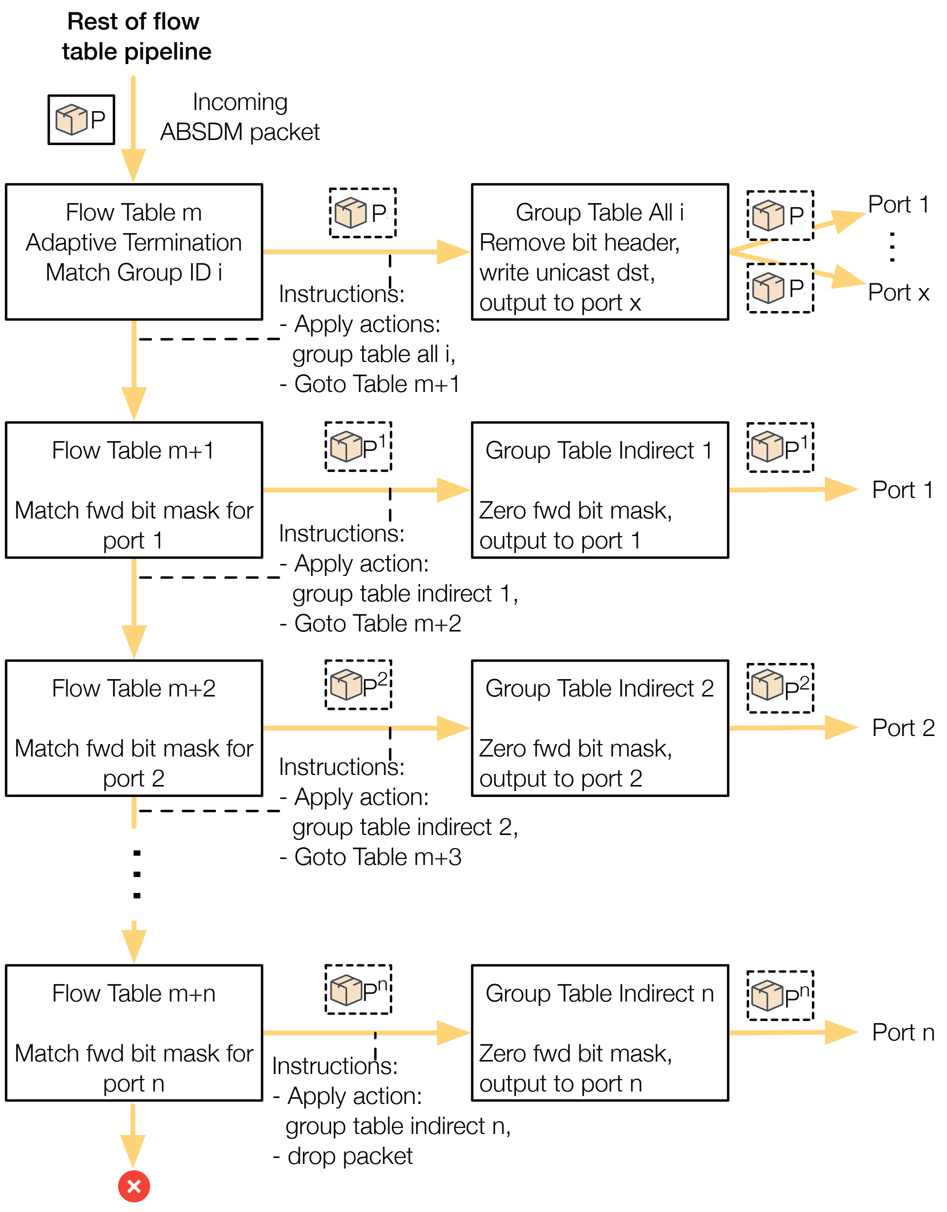

Rest of flow table pipeline
P
Incoming ABSDM packet
Flow Table m Adaptive Termination Match Group ID i
P
Instructions:
- Apply actions: group table all i,
- Goto Table m+1
Group Table All i Remove bit header, write unicast dst, output to port x
P
P
Port 1
Port x
Flow Table m+1
Match fwd bit mask for port 1
P1
Instructions:
- Apply action: group table indirect 1,
- Goto Table m+2
Group Table Indirect 1
Zero fwd bit mask, output to port 1
P1
Port 1
Flow Table m+2
Match fwd bit mask for port 2
P2
Instructions:
- Apply action: group table indirect 2,
- Goto Table m+3
Group Table Indirect 2
Zero fwd bit mask, output to port 2
P2
Port 2
Flow Table m+n
Match fwd bit mask for port n
Pn
Instructions:
- Apply action: group table indirect n,
- drop packet
Group Table Indirect n
Zero fwd bit mask, output to port n
Pn
Port n

Figure 5.12: The ABSDM OpenFlow pipeline

we assumed OpenFlow to support the specific bit header format used by ABSDM. This could be implemented, e.g., using the OpenFlow experimenter feature. If the header format is not supported, the 128-bit IPv6 destination address could be used to identify bit indexed multicast traffic and to encode the bit field. The 20-bit flow label header field of IPv6 Group identification could be used for group identification.

Table 5.4: Requirements for the ABSDM forwarding pipeline

Resource/Feature	**Consumption**
Flow tables	Num. of next hops + 1
Num. group tables type *indirect*	Num. of next hops
Num. group tables type *all*	Num. of groups with local unicast conversion
Num. of flow entries	Bit string length

Table 5.5: Required OpenFlow features over using version 1.0 as baseline (adapted from [Wel16]).

Feature	**Min. OpenFlow version**	**Comment**
Group Tables	1.1	Per next-hop packet processing
Match fields with arbitrary bit masks	1.2	Matching single bits of BIER destination addresses
Set field values with arbitrary bit mask	1.5	Zero current forwarding bit mask

Table m terminates packets addressed directly to the switch and enables early duplication strategies for adaptiveness, both of which require the conversion of traffic from multicast to unicast. In this table, entries can be added that match for a specific multicast group. Then, per group, the packets can be converted to unicast and replicated again if required using an OpenFlow *all* table.

The functional requirements for OpenFlow devices to able to operate ABSDM are listed in Table 5.4. While the approach is expected to be very efficient regarding flow entry consumption, it requires the data plane element to support a high number of flow tables.

In addition to a flow table per port, the system requires some features that are only available in recent versions of OpenFlow. The features and the OpenFlow version they are available from are listed in Table 5.5.

An extended description of the BSDM approach including its application interface can be found in the master's thesis of P. Welzel [Wel16].

5.4.4 *Mitigating Global Matching Memory Shortages*

The two presented adaptive multicast approaches, ASDM and ABSDM, are both designed to being able to control the tradeoff between memory and data rate. ASDM has fewer requirements for data plane elements, is very flexible when it comes to deployments with partial SDN support as well as selective usage of data plane elements in general. ABSDM promises to be more efficient than ASDM but also has much higher requirements for data plane elements and requires the complete data plane to be SDN enabled. To run ABSDM, data plane elements need to support OpenFlow 1.5, and one flow table is required in the pipeline for every network port. At this time, only the devices from a single vendor are known to support this many flow tables. The goal is, therefore, to use ABSDM where possible and ASDM where required. Both approaches require the SDN switches to be able to replicate a large number of packets, modify their header before sending, and sending multiple replicas out the same port.

ASDM and ABSDM use at maximum one OpenFlow flow entry on each multicast switch per group. The maximum number of flow entries that different events in an adaptive multicast control plane application affect are listed in Table 5.6. Based on the resource discovery process conducted in Chapter 4 we identified relevant resources on the control path and found them to be the match table memory interfaces and the match table space. While the number of flow entries that are modified, added, or deleted by a single group or member event is small, the total number of groups is expected to be huge in comparison to the number of data plane elements in the network. The metrics listed in Table 5.6 are the same for both approaches. However, the number of switches with group state in ABSDM is less or equal than in ASDM.

Table 5.6: ASDM control path resource consumption.

Metric	**Event type**	**Max. num. of affected flow entries**	
		ASDM	**ABSDM**
Member events	Add	2	2
	Delete	2	2
	Modify	1	1
Group events	Add	1	1
	Delete	(Num. of switches with group state)	(Num. of switches with group state)

Both forwarding methods consume their resources in both the core and the edge area of ISP networks. Resource bottlenecks and shortages in both areas can affect the reliability of ASDM and ABSDM control plane application. Detecting bottlenecks before actual errors occur, e.g., because there is no match table space left on a device is crucial for predictable behavior.

To that end, the applications continuously keep statistics on their consumption of the match table space and flow table memory interface resources, both by network area and by device. Both statistics need input from the controller since applications with higher priority might consume these resources as well. The applications establish two resource limits with the controller: the minimum resource requirements and the maximum resource requirements. The maximum requirements are expected to be indirectly provided by the network operator by providing the maximum design load for the multicast system regarding multicast groups and members. If this maximum load is exceeded, the multicast application may deny additional service requests. Given this load, the control plane application derives its expected resource requirements, regarding flow table space, flow table memory interface, and data rate reduction. The data rate reduction estimate provides, as a ratio of the estimated unicast data rate, how much network capacity the system is going to save. The minimum resource requirements are derived by taking the current service load of the system and adding a trend of the recent past and deriving the resource requirements for this load. Therefore, the minimum resource requirements specify the resources the system needs to stay operational in the short term, while the maximum resource requirements specify the design load.

If the controller detects that one the requirements might be missed, the control plane application is informed of this. The first action in both cases is to decide which resource might fail and which of the requirements are failed, the minimum, the maximum or both. We will focus on the flow table space resource in detail in the remainder of this section.

In case the minimum requirement for the flow table space resource is missed, the control plane application investigates its course of action. The primary option in the case is to increase the unicast conversion threshold. Based on the current workload, the application simulates the impact of the modification of the threshold. It then selects the smallest value that would bring the resource consumption back in line with the available resources. Then, the application queries the controller if the corresponding increase in data rate is acceptable. Therefore, the controller needs to be able to get up to date information on the data rate of existing multicast flows, as well as the other traffic in the network. We assume a system is available to provide estimates of the data rates in the network, e.g., based on the resource-efficient designs by Moshref et al. [Mos+14] and Hark et al. [Har+16]. If the increase in data rates is acceptable, the multicast application proceeds to switch the unicast conversion threshold to its new value. At first, it is applied to new groups. Then, gradually all existing groups are switched to the new value as well. At this point, the resource requirements should be met again.

If modifying the unicast conversion threshold proves not to be sufficient to mitigate the resource shortage, the application sends a notification to the operator and switches to a degraded state. In this state, new groups are not accepted, and group members are only accepted if the resources are available to support them. Thereby, the application remains operational and responsive even in a case when the data plane does not provide sufficient resources.

In case the maximum resource requirements are not met a new maximum load is derived based on the available resources and the operator is informed about this new limit. When the actual load closes in to the maximum load, the system reacts in the same manner as in the case when the minimum resource requirements cannot be met.

Detecting and mitigating resource requirements not only requires control plane applications to foresee and prepare for shortages and bottleneck, but also the controller to have complete knowledge of the network. Estimating the remaining link and flow table space capacities in a large network are, therefore, a requirement for reliable SDN controllers in ISP networks.

5.5 DISCUSSION

In this investigation of the design of mitigation strategies of two representative control plane applications, we gained an insight into their information requirements.

Our designs provide well-defined approaches to mitigating control path bottlenecks. The network function chaining application introduces a simple yet efficient approach to managing users and their function chains in ISP networks. The drawback is a potentially large number of flow entries that must be moved in case of a device failure. We mitigate this effect by introducing a design to share the flow entry installation load between two SDN devices, a hardware, and a software switch. Furthermore, we provide a clear decision criterion on how to share the load. The adaptive multicast approach introduces the ability to mitigate global flow entry capacity shortages by adjusting the data-rate-state tradeoff if multicasting. We apply it to operate on a unicast substrate forwarding underlay as well as to a highly efficient bit-indexed replication forwarding underlay. In both approaches, the tradeoff is controlled by a single parameter.

For both applications, we found that renewable resources, both in the control- and in the data path, require timely and accurate information. This puts high monitoring demands on the controller that must provide this information to a potentially large number of applications for a large number of data plane elements. Furthermore, in both cases, the application must specify its resource requirements. The controller must provide information if these can be met, if not a resource event is triggered. However, the timeframes in which this information is required are quite different. The network function chaining application requests resources in case of a failover event and requires an answer as fast as possible during the whole process. The adaptive multicast application registers its long-term resource requirements and expects events in case these requirements might not be met.

We found that in both cases, the controller must provide suitable alternatives for the mitigation approaches. The network function chaining application needs an SDN device with specific characteristics; namely it must be adjacent to the overloaded data plane element. The adaptive multicast approach increases the data rate of its traffic in the network, which requires the controller to be able to determine if the additional load can be handled.

In conclusion, the presented approaches promise to mitigate the investigated resource events. At the same time, we found new requirements posed to SDN controllers to support the mitigation process.

6 EVALUATING MITIGATION APPROACHES FOR CONTROL PATH BOTTLENECKS

Our designs to mitigate control path bottlenecks presented in Chapter 5 are evaluated in this chapter. Both control plane applications, network function chaining, and ASDM are investigated in an experimental setting, either in a testbed or an emulated network. However, since the two applications apply to different areas of ISP networks, for both, first the workload and experimental setting are introduced separately before presenting the prototypical implementations of the systems and the results. The evaluation results of the mitigation approach for the network function chaining application is presented in Section 6.1, the results of the evaluation of our ASDM design in Section 6.2. The results and their consequences for the design of the interface between SDN controllers and applications are discussed in Section 6.3.

The investigation in this chapter partially relies on input from two papers [Ble+15a; Ble+15b] and two supervised student theses by S. Bleidner and T. Volk [Ble15; Vol14].

6.1 EVALUATION OF THE MITIGATION OF LOCAL FLOW-UPDATE RESOURCE BOTTLENECKS

The goal of this evaluation is to determine which information the network function chaining control plane application should be used to decide if and how much of the flow entry addition load should be shared between virtual switch on the NFV infrastructure server and the adjacent ToR switch. To that end, we investigate two different approaches: using information from the SDN controller that infers performance information for two data plane elements separately and emulating the actual failover event by the control plane application.

The investigated scenario and the used testbed are described in Section 6.1.1. The workload is derived and discussed in Section 6.1.2 as well as the prototype control plane application. The results are presented in Section 6.1.3 followed by a discussion on what information control plane applications should to base their decisions on in Section 6.1.4.

6.1.1 *Scenario and Testbed*

As introduced in Section 5.2.1 the scenario of the control path resource bottleneck is the failure of an NFV infrastructure server and the subsequent need to move affected network function chain instances to a new server. The specific scenario that is investigated is depicted in Figure 6.1. Before the failure event, two servers host two different network function chains with a large number of function chain instances. Users in Group A use network function chain A (Chain A) the users in Group B use Chain B. Chain A consists of four VNFs, types 1-4, and Chain B uses VNF types 5-8. These last four VNF types are hosted only on NFV infrastructure server 2 (Server 2), while the first four types 1-4 can be hosted on both NFV infrastructure servers, Server 1 and Server 2. Three of the VNF types per chain are assumed to use shared instances, while one VNF of each chain uses dedicated instances. For the dedicated VNF instances, one instance is created for every subscriber in the two groups. Figure 6.1 depicts the VNF types only and abstracts from the individual instances.

In the scenario, Server 1 crashes at some point, which means that the instances of the four VNFs of Chain A that is used by Group A must be moved to Server 2, including the network function chain instances. In this investigation, we focus on the installation of new OpenFlow flow entries and assume the VNF instances are created through one of the approaches presented in the design of the system in Section 5.3. The new path of the instances of Chain A after the failover to Server 2 is depicted by the dashed part of the Chain A. This leads to a peak load of OpenFlow flow entry additions on Virtual Switch 2, which is mitigated with our approach by sharing the load with ToR Switch 1.

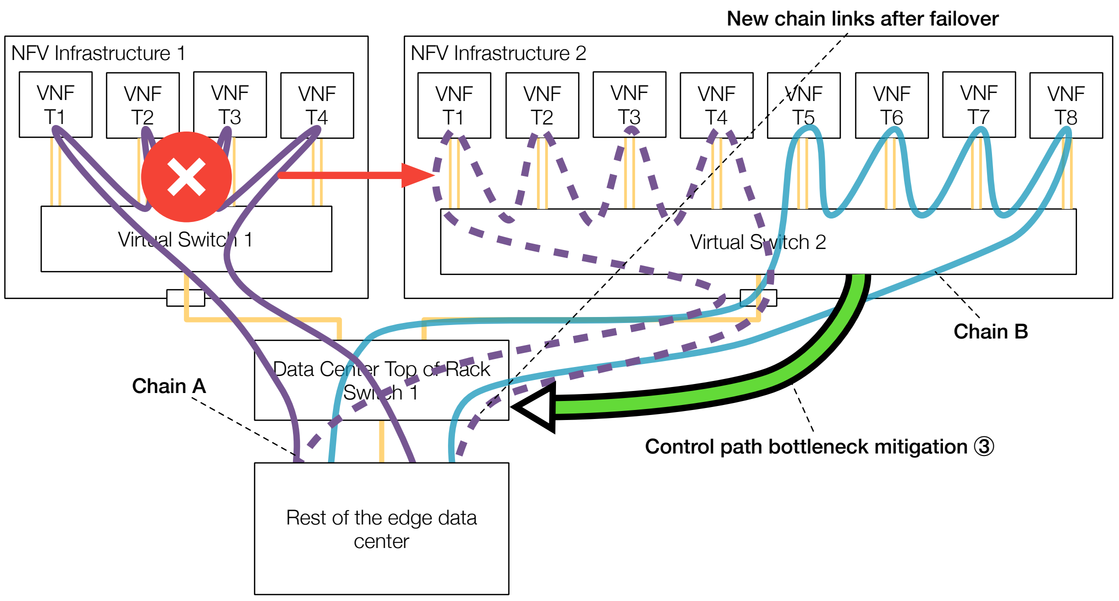


Figure 6.1: The evaluated network function chaining scenario (adapted from [Ble+15a]).

This evaluation focusses on the control path of the SDN data plane. Therefore, only the relevant parts of the scenario are implemented in the testbed. As depicted in Figure 6.2 the testbed includes the ToR switch as well as Server 2 from the scenario. Server 1 fails, hence, there is no need to emulate it in the testbed. Chain B and Group B are assumed to be a base load for Server 2 but are not expected to contribute to the load during the failure case. The VNF instances as well as their links are pure data path elements of the network and are not required for investigating the control path. Virtualization servers

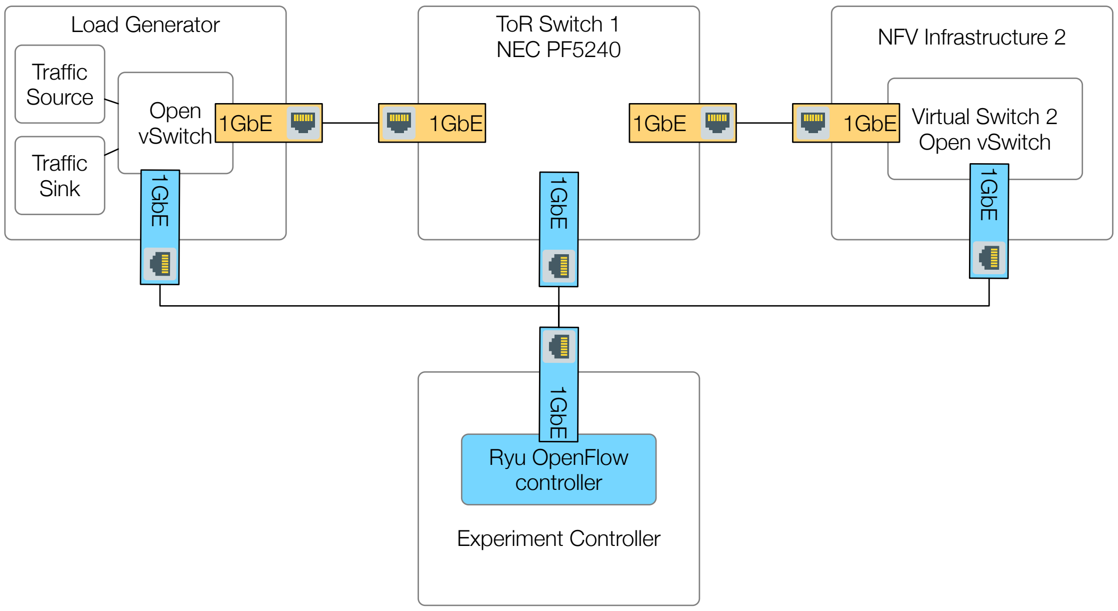


Figure 6.2: The implementation of the evaluated scenario in the testbed (adapted from [Ble+15a]).

today employ a range of isolation approaches to separate the load of VMs from the load of the virtual switch and the management functions [Ble+16a]. Hence, their load is replaced by using only the part of the server resources that are used for the SDN control path of the virtual switch.

In their experiments on a single virtualization server, Manco et al. [Man+17] use four CPU cores for virtual switching and management and 60 CPU cores for VMs. We conclude that the control path part of the resources used for virtual switching is equivalent to one CPU core. Therefore, the control path of Virtual Switch 2 is represented in our experiments by a single core CPU as listed in Table 6.1.

An NEC PF5240 switch represents the ToR switch. It has a maximum flow entry capacity of 160,000[1] entries, which is appropriate for the scale required in this investigation. In contrast to that, the Edge-Core AS5712-54X is a newer design and includes a state-of-the-art management system but can host only about 2,000 OpenFlow flow entries. The fact that the management system of the NEC PF5240 device is not state-of-the-art is not a drawback, because the control path bottleneck is expected to be the TCAM memory interface. Instead of the 10GbE interfaces expected by an actual deployment, the switch uses 1GbE interfaces. However, since the focus is on the control path, this has no impact on the results.

The measurements of the completion time of the failover are conducted using the OpenFlow barrier primitive. If requested by the controller, a data plane element sends a barrier_reply message to the controller once all previously received OpenFlow messages

1 NEC: NEC ProgrammableFlow UNIVERGE PF5240, Data sheet.

have been completely processed. Literature reports that on some devices, this primitive does not lead to accurate results. To that end, we measured the accuracy of the barrier primitive on both data plane elements and found that both are sufficiently accurate. The measurement results are available in the paper of Blendin et al. and the thesis of Bleidner [Ble+15a; Ble15]. The load generation and measurements are conducted on a dedicated server with an additional virtual switch that enables the efficient control of the experiment.

Finally, a dedicated server hosts the OpenFlow controller software. We found the accuracy of the receiving process of barriers messages of the OpenFlow controller lacking. Therefore, we collect and analyze packet traces of the control path traffic on the controller and thereby determine the barrier time. For more details on the testbed calibration process, refer to the thesis of Bleidner [Ble15].

Table 6.1: Testbed hardware specifications [Ble+15a].

Node	Hardware	Software
Control path of Virtual Switch 2	Intel Pentium G640 CPU, 8GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Open vSwitch 2.3.1
ToR Switch	NEC PF5240, 48 1GbE & 4 10GbE ports	Firmware version 5.1.1.0
Edge Switch & Users	Intel Pentium G640 CPU, 8GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Open vSwitch 2.3.1
Experiment Controller	Intel Core i5-3470 CPU, 32GB RAM, Intel 82579LM NIC	Ubuntu 12.04, Ryu 3.19

6.1.2 *Workload and Prototype*

Publicly available information on the data rates and load details in real-world access networks is scarce. The reason for that is that many ISPs consider this information business secrets. Available numbers are often aggregated for many subscribers without the required context information to determine the load on individual network components, e.g., as published by Pariag and Brecht [PB17]. An exception is the publication by Bulut and Szymanski [BS15] on mobile access networks. However, while the authors provide per-session statistics such as the maximum busy hour data rate, the authors do not disclose the origin or size of their data set. We will consider this publication when designing the workload but will still need to derive parts of it from a set of assumptions. To that end, we first establish upper bounds for the essential components in the system and then derive the workload parameters for the evaluation.

Following Manco et al. [Man+17] we assume that the NFV infrastructure server uses four CPU cores for virtual switching and management and 60 CPU cores for VMs. The virtual switch is expected to operate Open vSwitch [Pfa+15].

The maximum number of flow entries per VNF instance is provided in Table 5.2. For the specific workload investigated here, the total number of flow entries for the whole system can be derived from the flow entries per subscriber of both groups on Server 2 as provided in Equation (6.1).

$$\mathrm{N}_{E_{\mathrm{vS}}} = 2\,\Sigma_{s\ in\ \mathrm{subscribers}} \left(1 + \left(\mathrm{Num.\ VNFI}^{s}_{\mathrm{local}}\right)\right) \tag{6.1}$$

where:

N = Total number of flow entry addition operations

E_{vS} = Data plane element: virtual switch

Bulut and Szymanski [BS15] state that the maximum busy hour data rate per subscriber for an undisclosed mobile operator in 2015 was 10kbit/s. Given the dramatic increase in 4G deployments, we expect this number to have increased in the past years. Therefore, a peak average data rate per subscriber of 100kbit/s is assumed in this evaluation. As stated in the scenario introduction, one VNF per chain uses dedicated instances per subscriber, while the others are shared between the subscribers.

Emmerich et al. [Emm+15] investigated the maximum viable flow entry set size for the state-of-the-art virtual switch software Open vSwitch. They concluded that the level three cache of the processor cores that run the virtual switch is a major limiting factor for the size of a flow entry set and showed that 14,500 flow entries are the upper limit of flows per core. Assuming the virtual switch operates on four cores, this yields a maximum of 58,000 flow entries. Open vSwitch uses a sophisticated flow entry caching mechanism that enables it to keep only those flow entries in memory that are used. Given that not all users are active at the same time, 60,000 flow entries are a natural upper bound for the network function chaining flow entries on the virtual switch.

Because of the disaggregated NFV architecture assumed for this investigation, dedicated VNF instance and shared VNF instances are used to create the network service required by the subscribers. Both network function chains, Chain A and Chain B, rely on four VNFs each, with one using dedicated instances per chain instances and three using VNF instances that are shared between chain instances. Server 2 is operated at less than half of its capacity, to be able to accept the same number of VNF instances in case a neighboring server fails.

Given that 60,000 is the highest viable flow entry capacity of the virtual switch, we can use Equation (6.2) to derive the corresponding number of subscribers per server, which is 6,000. For the given configuration of VNFs and 6,000 subscribers, applying the equation results in $6,000$ dedicated instances $+ 3$ shared instances $= 6,003$ instances.

Minimal VM designs as proposed by Manco et al. [Man+17] were demonstrated to being able to host up to 8,000 VMs on 60 CPU cores. Traditional container technology was shown to run up to 3,000 VNF instances on the same server. Therefore, hosting three traditional VMs or containers in addition to 6,000 minimal VMs seems viable.

The number of times the virtual switch handles each packet can be derived by adding up the number of chain links into, within, and out of the server. This number equals the number of the OpenFlow flow entries per direction per subscriber. We multiply this number with the average download data rate per subscriber to obtain the total data rate for the virtual switch.

$$\mathrm{D}_{E_{\mathrm{vS}}} = d \, \Sigma_{s \; in \; \text{subscribers}} \left(1 + \left(\text{Num. VNFI}^{\text{s}}_{\text{local}}\right)\right) \tag{6.2}$$

$$\tag{6.3}$$

where:

$D \; = $ Total data rate

$d \; = $ Average data rate per subscriber

$E_{\mathrm{vS}} = $ Data plane element: virtual switch

With the assumed average peak data rate per subscriber of 0.1Mbit/s, this results in a total throughput for the virtual switch of 3000Mbit/s. The data rate in and out of the NFV infrastructure server is determined by the number of chain links entering and leaving the switch. In this scenario, the data rate is 2 directions $*$ $6,000$ subscribers $*$ 0.1Mbit/s average peak data rate $= 1,200$Mbit/s. Both values are viable using existing technology.

We conclude the derivation of the workload parameters for the evaluation by listing them in Table 6.2.

Table 6.2: Parameter values used in the network function chaining evaluation [Ble+15a].

Parameter	**Investigated Values**			
Subscribers in the network function chaining system	1,000	2,000	5,000	6,000
Subscribers in Group A that are affected by the node failure	500	1,000	2,500	3,000
Total number of flow entries installed during failover	5,000	10,000	25,000	30,000
Total number of flow entries installed during failover and eligible for being moved to the ToR switch	1,000	2,000	5,000	6,000
% of total number of flow entries installed during failover and eligible for being moved and moved to the ToR switch	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55			

The prototype for investigating the flow entry addition load sharing was implemented using the Ryu[2] OpenFlow controller. Its main task is to generate the flow entry addition load, to share it between the two OpenFlow switches, and to measure the completion time. The NEC PF5240 has specific requirements in which OpenFlow flow table the entries are installed, while the Open vSwitch does not pose any such requirements. The decision logic we are investigating in this evaluation is not implemented as part

2 https://osrg.github.io/ryu/, [Accessed September 18th, 2018]

of the control plane application but as a part of the experimentation controller. All installed flow entries are exact matches and have the same priority. More details on the implementation can be found in the thesis of Bleidner [Ble15].

6.1.3 *Results*

The goal of this investigation is to understand how the network function chaining application should determine if load sharing should be used and to what extent. We investigate two different approaches: using inferred control path performance data from the controller and emulating the actual failover event.

For the inferred control path information both devices, the Open vSwitch virtual switch and the NEC PF5240 ToR switch, are measured separately. The detailed resource topology is not required for this specific measurement. This is because the network function chaining application is considered the only one that operates at the highest priority in the control plane. Therefore, when using the virtualization approach presented in Chapter 4, there is no control path interference from other applications.

We measure the addition performance for flow entries by a set of experiments with gradually increasing the number of flow entries, with 20 repetitions for each experiment. The approach is chosen to capture the effect of TCAM-based match table, where the flow entry addition performance decreases with increasing table space utilization. All flow entries in this experiment have the same priority. Nevertheless, we measure both the addition of flows with the same priority and with different priorities. Even though the exact memory type used by the hardware is not known and could be TCAM, BCAM, or event SRAM-based, we aim to capture the effect of priorities in case TCAM is used. The flow entry addition performance of the complete range of flow entry numbers from 1,000 to 30,000 is investigated. The resulting performance for both switches is depicted in Figure 6.3. In addition to that, for each experiment, with different or same priorities, a performance model fitted. The model is created using local polynomial regression fitting and the LOESS method [RLANG].

The performance for adding flow entries with the same priority on the NEC PF5240 peaks at around 800 additions per second at 1,000 total entries added. It slowly decreases to about 740 additions per second at 29,000 total added entries. As expected from literature, adding flow entries with different priorities is slower, about 680 at 1,000 total flow entries. Furthermore, it decreases faster to about 290 additions at 29,000 total added entries.

In contrast to that, Open vSwitch shows similar behavior in both experiments. Both experiments start around 7,500 additions per second at 1,000 total added entries. Interestingly, the performance increases with the total number of added entries in each experiment. In both experiments the performance peaks at about 10,000 additions per second when adding 30,000 flow entries in one run. Using these numbers, selecting the number of flow entries to move to the hardware switch can easily be done.

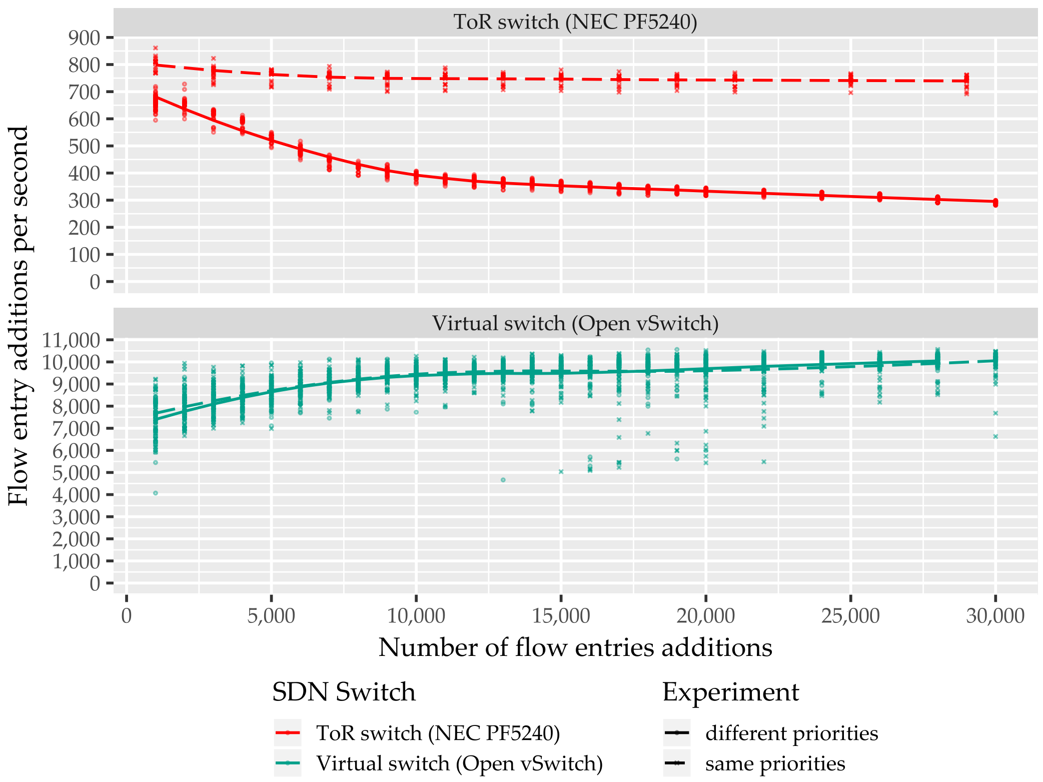


Figure 6.3: Flow entry addition performance for Open vSwitch and the NEC PF5240 (adapted from [Ble+15a]).

The results of the investigation of the emulated failover scenario are depicted in Figure 6.4. Due to the more complex scenario, the plots are structured differently. The results for the failover scenario is depicted with a different number of subscribers that are affected by the failure, ranging from 500 to 3000. The actual number of flow entries added per switch are not depicted in the figures but are listed in Table 6.2. Instead, the crucial failover completion time depicted, which is the metric the system aims to minimize. For small numbers of affected subscribers up to 40% of the eligible flow entries can be moved to the ToR switch before increasing the total failover completion time. This number shrinks to 20% for the scenario with 3,000 affected subscribers.

A model for the completion time behavior of both switches is derived for the data from the failover experiment using the LOESS approach. For comparison, the models that are inferred from the individual performance modeling experiments are depicted as well as dashed lines. Where the models can predict results, they are significantly lower than the results from the failover experiment. Interestingly, the difference in the results for the NEC PF5240 is about two times as big as the ones for Open vSwitch.

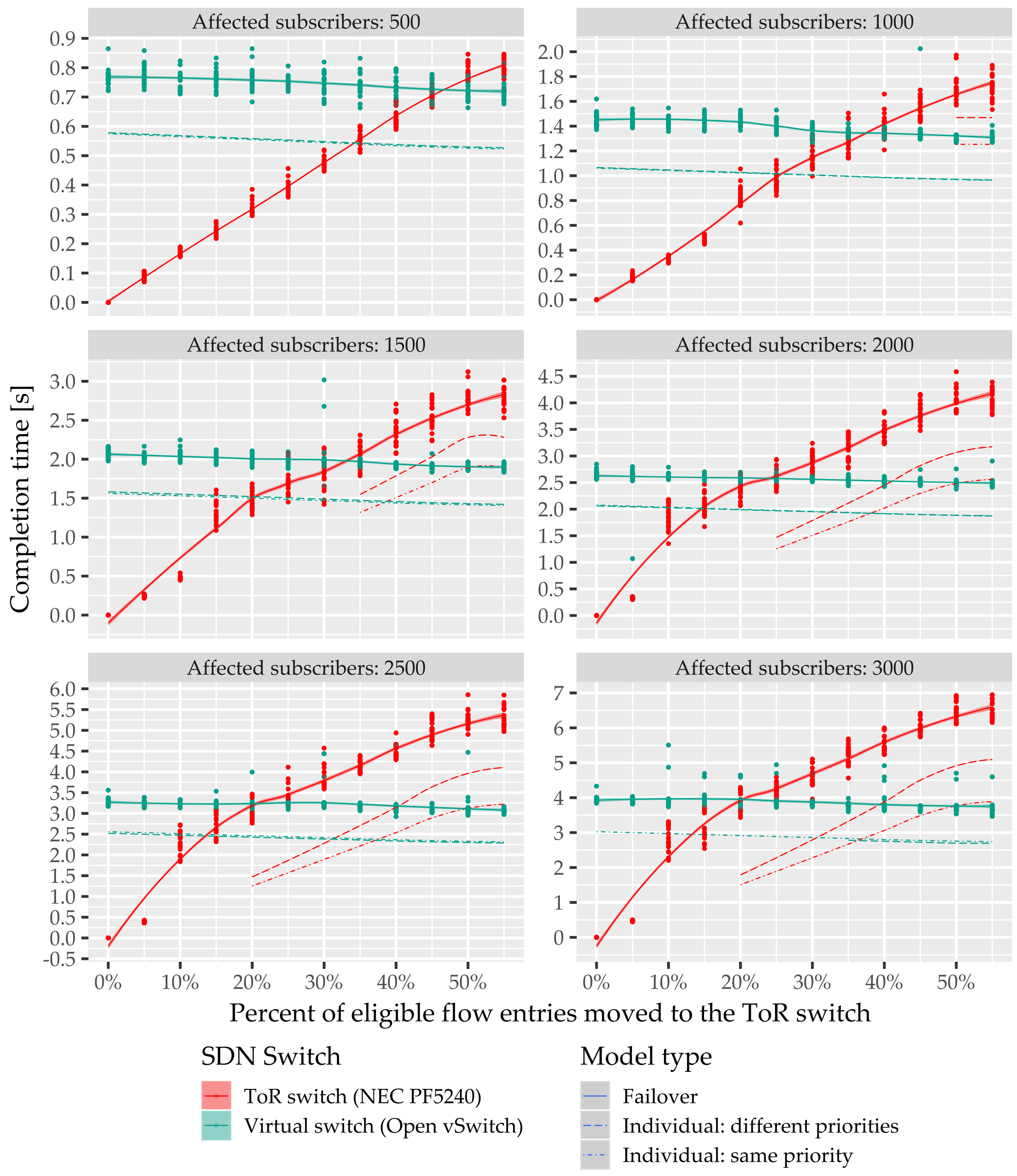


Figure 6.4: Failover completion time for the control path bottleneck mitigation approach (adapted from [Ble15])

6.1.4 *Discussion of the Evaluation Results*

The results clearly show that the performance difference in adding flow entries between the two data plane elements is too large to have a significant effect on the failover completion time. However, the fact that the software switch is an order of magnitude faster when adding flow entries is surprising. Huang et al. [HYS] report only about 400 flow additions per second on an Open vSwitch. We explain this performance difference by optimizations of the Open vSwitch software between version 1.7 that was used by Huang et al. in their experiments and version 2.3 used in our experiments.

The difference between the performance models created using separate measurements of the individual switches and the ones created using the failover experiment is significant. This shows that while the performance of the data plane elements' control path is important, it is not sufficient for performance predictions. Instead, the complete control path between the control plane application and the data plane elements, including the controller, must be taken into account. We, therefore, conclude that performance-related scenarios should be tested directly by the applications under realistic conditions instead of relying on performance numbers inferred by the controller.

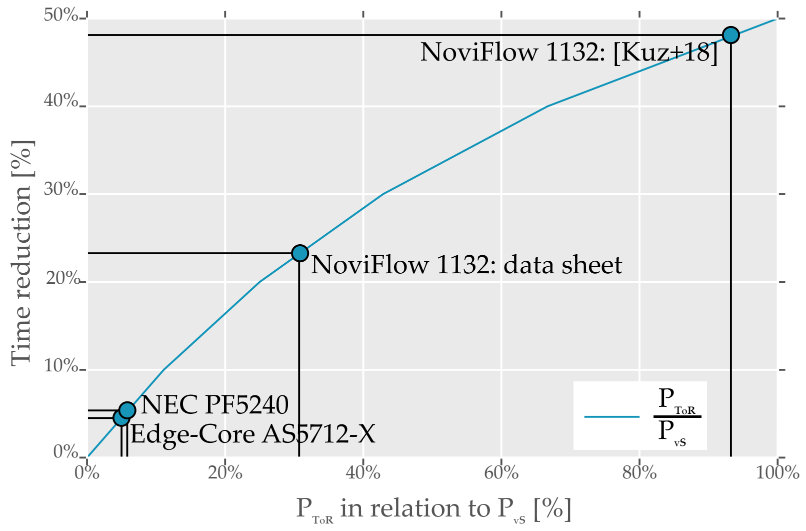


Figure 6.5: Completion time reduction overview (adapted from [Ble+15a])

Finally, we investigate which kind of device might offer a flow entry addition performance sufficient to reduce the failover completion time significantly. The estimated connection between the performance ratio of the virtual switch and the ToR switch with the ratio of eligible flow entries taken into consideration is depicted in Figure 6.5. For Open vSwitch, we use 7,500 flow entry additions per second as measured in the failover experiment as the reference value; for the NEC PF5240 the measured value is used as well. To get an estimate, we compare these values with values from the literature. One switch with a high flow entry addition performance rated at 2,500 per second in its data sheet[3] and the update rate measured to be about 7,000 by Kuzniar et a. [Kuz+18] is the NoviFlow NoviSwitch 1132. It is based on an EZchip/Mellanox NP-4 NPU, which offers only 100Gbit/s of data path throughput, but an excellent control path performance.

3 NoviFlow: NoviSwitch 1132 High Performance OpenFlow Switch, DS2017-NS1132-400-03, Data sheet, 2013

We conclude that this type of switch could be deployed at locations where they could contribute most to mitigate flow entry addition bottlenecks. The two devices investigated in this paper, the NEC PF5240 and the Edge-Core AS5712-54X, offer both deficient performance for adding flow entries and cannot contribute to mitigating control path bottlenecks in this scenario.

6.2 EVALUATION OF THE MITIGATION OF GLOBAL RESOURCE SHORTAGES

We investigate the ability of ASDM to mitigate global resource shortages in this section. The goals and metrics we use to do so are introduced in Section 6.2.1 followed by the presentation of the scenario and workload in Section 6.2.2. The evaluation results are described in Section 6.2.3 and discussed in Section 6.2.4.

6.2.1 *Goals and Metrics*

The goal of this evaluation is to determine the control range of the adaptivity mechanism used in ASDM and ABSDM as well as to determine what information is required to decide on how to conduct the adaptation process. In both cases, the unicast conversion threshold, designed to be the only parameter of the system is critical. The controller, as well as the applications, need to be able to predict the effect of changing the unicast conversion threshold to ensure that the planned reduction in flow entries is achieved and, at the same time, the corresponding increase in data rates is acceptable for the network.

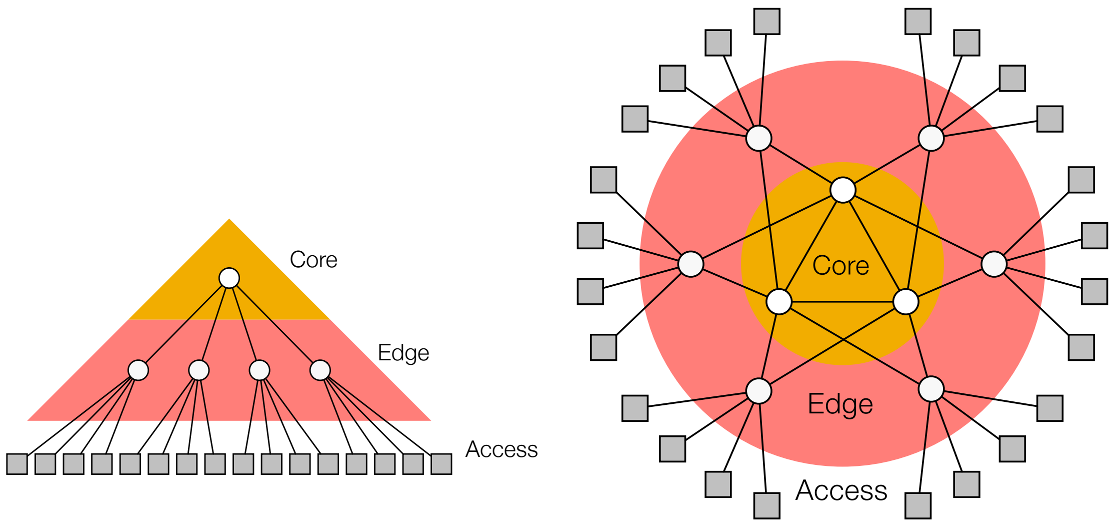
The adaptive multicasting system is designed to be deployed in a wide range of network topologies and use cases. Therefore, the influence of two different topologies on the system and its adaptivity characteristics are investigated. Furthermore, in the decision process, the effects on the different areas of an ISP network must be investigated to determine if they should to be treated differently.

The investigation is conducted in an emulated network environment instead of using a simulator. This is done to ensure that the proposed approach can be implemented with SDN today and because, to the best knowledge of the author, no simulator for large-scale SDN-based ISP networks is available today.

6.2.2 *Scenario and Workload*

ISPs are expected to have millions of customers, a vast geographically distributed core network and hundreds or even thousands of edge data centers. Investigating the SDN control plane applications at these scales is difficult. A sufficiently large testbed with hardware devices is not available, which is why the Mininet network emulator published by Lantz et al. [LHM10] is used. MaxiNet by Wette et al. [Wet+14] is an improved version of Mininet that supports running emulations distributed on multiple servers to improve the scale of emulated network experiments. However, even MaxiNet is not enabling the emulated the network sizes required to mimic actual ISP networks. Therefore, we use Mininet and conduct representative small-scale experiments, the results of which we expect to provide insights into corresponding large-scale experiments.

ASDM is investigated on the example of two different network topologies, a tree topology and an idealized ISP topology as depicted in Figure 6.6. The tree topology represents the best-case scenario for multicast, with an optimal efficiency gain over unicast. The results for the tree topology provide an insight into the effectiveness of the adaptation approach in an optimal use case for multicast. The results of the ISP topology should provide an exemplary result for ISP networks in general. Public data of real ISP network topologies are available on the Internet Topology Zoo by Knight et al. [Kni+11]. However, the data set is focused on the core network sections and does not provide insight into the aggregation sections, which are relevant for this work. Therefore, we choose an ISP topology that is modeled after a large European ISP [DG06]. It has been extensively used in literature to investigate SDM [Ble13; RBH15; Rüc+16].



(a) Tree topology. (b) ISP topology.

Figure 6.6: Investigated topologies (adapted from [Ble+15b]).

Both topology models include the relevant areas of ISP networks, *core*, *edge*, and *access*. The *access* area represents the traffic recipients in our topologies and is not controlled by the ISP control plane. The *edge* area is controlled by the control plane in all scenarios, while in the *core* area SDN control is optional. We, therefore, investigate the ISP topology twice, once with SDN control of the *core* area, and once without, termed *unicast core*. The *unicast core* is included in this investigation because the *core* area is expected to be the last areas of ISP networks that become SDN enabled, due to its importance.

Table 6.3: Investigated network topologies (adapted from [Ble+15b]).

	Tree topology		Triangle topology	
Zone	**# Nodes**	**Fanout**	**# Nodes**	**Fanout**
Level 1 / Inner Core	1	4	3	2
Level 2 / Outer Core	4	4	6	4
Level 3 / Inner Edge	16	4	24	4
Level 4 / Outer Edge	64	16	96	11
Total number of data plane elements	85	-	129	-
Hosts	1024	-	1056	-

We investigate the topologies at the largest scale that the testbed allows us to. The corresponding parameters are listed in Table 6.3. The scenario is assumed to be the daily traffic peak in the evening as reported by Maier et al. [Mai+09]. We expect half of the available ~1024 hosts to actively access a multicast-able content, with each host being a member of at maximum one multicast group. The group sizes are distributed in a Zipf-like manner through a random process where the ingress node for groups is always placed in the edge area of the network. An overview of the group characteristics used in the evaluation and the investigated unicast conversion threshold values are listed in Table 6.4.

Table 6.4: Multicast group characteristics and parameter values used in the evaluation [Ble+15b].

Characteristics	Value
Num. of multicast groups	252
Total number of group members	3072
Number of different group sizes	6
Group size distribution (num. $*$ members)	$4 * 128, 8 * 64, 16 * 32, 32 * 16, 64 * 8, 128 * 4$
Unicast conversion threshold T	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 32, 64, 128

Since there are no dependencies between different multicast groups, we conduct experiments for each group size separately, which results in a total of 3072 active group members per experiment. Each group configuration is repeated 30 times on the emulated network, with a newly created, random set of multicast groups in each repetition.

Multicast traffic is emulated by transmitting a 128 kbit/s audio file per group. Using high data rate streams, such as IPTV is unfeasible in the emulated environment because of its performance constraints. 128 kbit/s is a popular data rate from streams, which represents web radio or audio streaming in general, which is expected to be a substantial part of live multicast traffic if multicast was available as an Internet-wide service today.

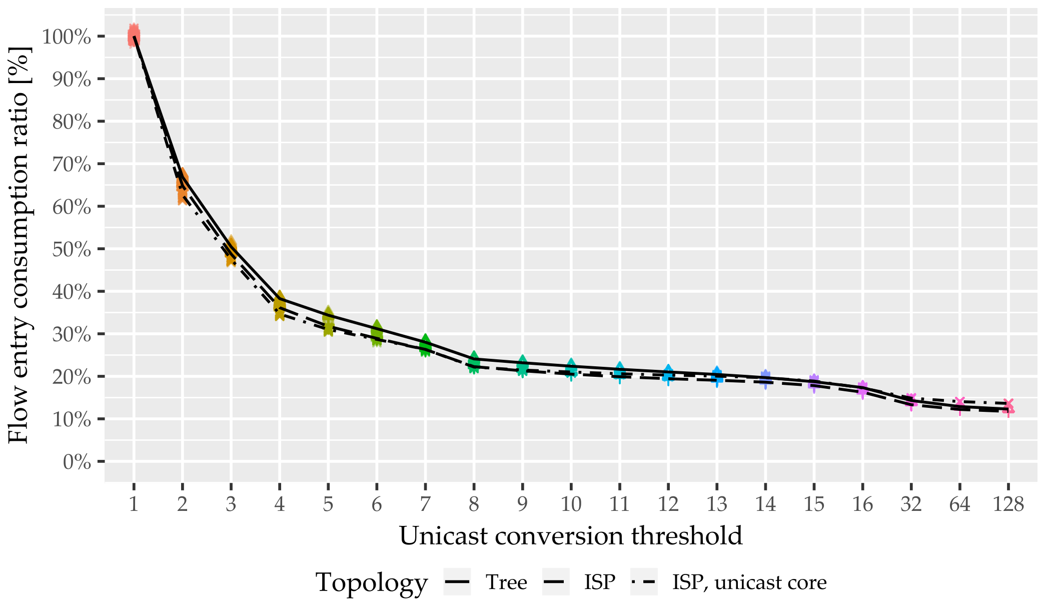


Figure 6.7: Impact of the unicast conversion threshold on the relative flow entry consumption (adapted from [Ble+15b]).

The investigated ASDM prototype is implemented using the Ryu[4] controller using OpenFlow version 1.3. The Mininet network emulator version 2.0 executes the experiments using on Open vSwitch to emulate SDN switches running on the Ubuntu 14.04 Linux operating system. The whole emulation is conducted in a VM with four CPU cores and 6GB memory on a server with an Intel Xeon E5-1410 processor.

6.2.3 *Results*

We first investigate the control range the unicast conversion threshold gives us over the flow entry consumption in the data plane. Figure 6.7 depicts the relative flow entry consumption for the measured scenario with the unicast conversion threshold as the independent variable. The results show that the flow entry consumption can be controlled in the range of the state consumption of the traditional late replication strategy denoted as 100% and less than 15% of this consumption for earlier duplication strategies. The traditional late replication strategy is selected by setting the unicast conversion threshold to $T = 1$. The reduced multicast state is chosen by setting the unicast conversion threshold at least $T = 32$. Unicast conversion threshold values between 1 and 8 show the most significant impact, which gradually shrinks for larger values. The differences between the investigated topologies are small. Therefore, we considered their impact on the state consumption behavior, i.e., the number flow entry

4 https://osrg.github.io/ryu/, [Accessed September 18th, 2018]

used in the network, not significant. This topology-independence is a significant result because it shows that the adaptation of the flow entry consumption is expected to not depend significantly on the topology, which makes its use easier for different network topologies.

The tradeoff between the state consumption and the data rate is essential to understand the impact of changing the unicast conversion threshold on data path resources. The connection between the data rate and the state consumption for different unicast conversion threshold values, termed data-rate-state profile, is depicted in Figure 6.8. Please note that the horizontal axis starts at 35% of the unicast data rate. While we found before that the flow entry consumption is independent of the investigated topologies, the data rate differs significantly between them. Not surprisingly, the tree topology gains the most from late replication strategies, e.g., when using $T = 1$. The impact of using multicast in the core network is visible but smaller. For unicast conversion thresholds larger than seven, the differences between the topologies diminish. The behavior is not surprising because, with larger thresholds, the depths of the topologies, and the group sizes both become small relative to the unicast conversion threshold. Therefore, the unicast conversion moves closer to the ingress switch, which reduces the impact of the topologies on the replication process.

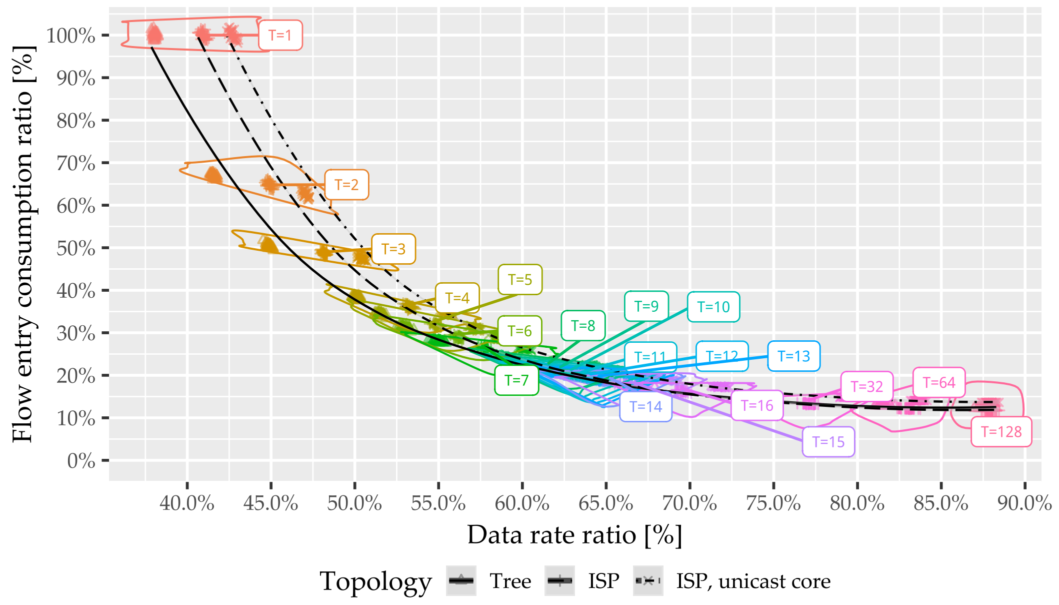


Figure 6.8: The data-rate-state profiles of the investigated topologies (adapted from [Ble+15b]).

Figure 6.9 gives a view on the adaptation choice as well as the data rate and state tradeoff in the ISP topology. Changing the threshold allows the control plane application and its operators to select a point close to the approximated profile lines. For example, when moving from $T = 3$ to $T = 4$, the consumption of flow entries is reduced from

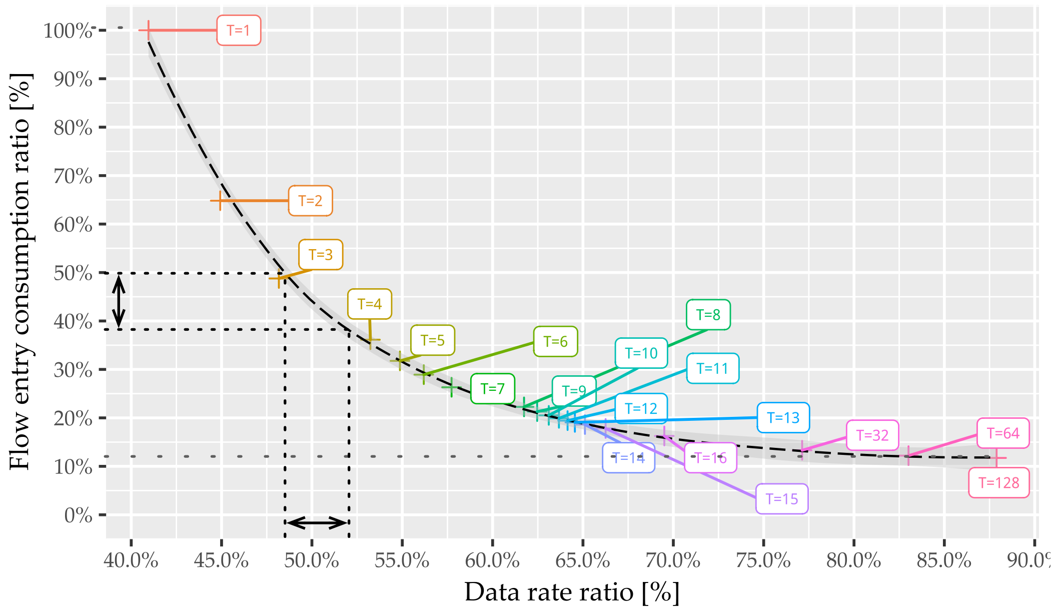


Figure 6.9: A detailed view on the tradeoff between data rate and state in the ISP topology (adapted from [Ble+15b])

about 49% to 36% compared to the late duplication strategy. At the same time, the relative data rate increases from 48% to 53%.

A switch from $T = 1$ to $T = 16$ increases the multicast traffic significantly, from 41% unicast data rate to 70% unicast data rate, or a 70% increase in data rate. Depending on the total data rate managed by the ASDM system, this might have a significant impact on the traffic engineering in the network. Multicast traffic is expected to be UDP-based, constant rate and well controllable. Hence, the typical burstiness and self-similarity patterns caused by per-connection feedback loops like those used by TCP [Wil+97] are not expected. Its constant bit-rate nature is expected to affect bursty traffic if queueing occurs. We, therefore, conclude that measuring the data rate of the multicast traffic in the ASDM is crucial. Furthermore, modifying the unicast conversion threshold should be done incrementally.

The results discussed before provide a clear understanding of the effect of the unicast conversion threshold on both, the data rate, and the state consumption from a global perspective. While the global perspective is important, and the system can mitigate local resource bottlenecks by skipping a specific data plane element, it is nevertheless important to understand the effects of the system on the resource consumption of individual data plane elements. To that end, Figure 6.10 provides an insight into the effect of the unicast conversion threshold on the peak state consumption on a single data plane element per area. The bars in the figure provide the absolute flow entry consumption per group member as denoted by the left vertical axis. The lines provide the

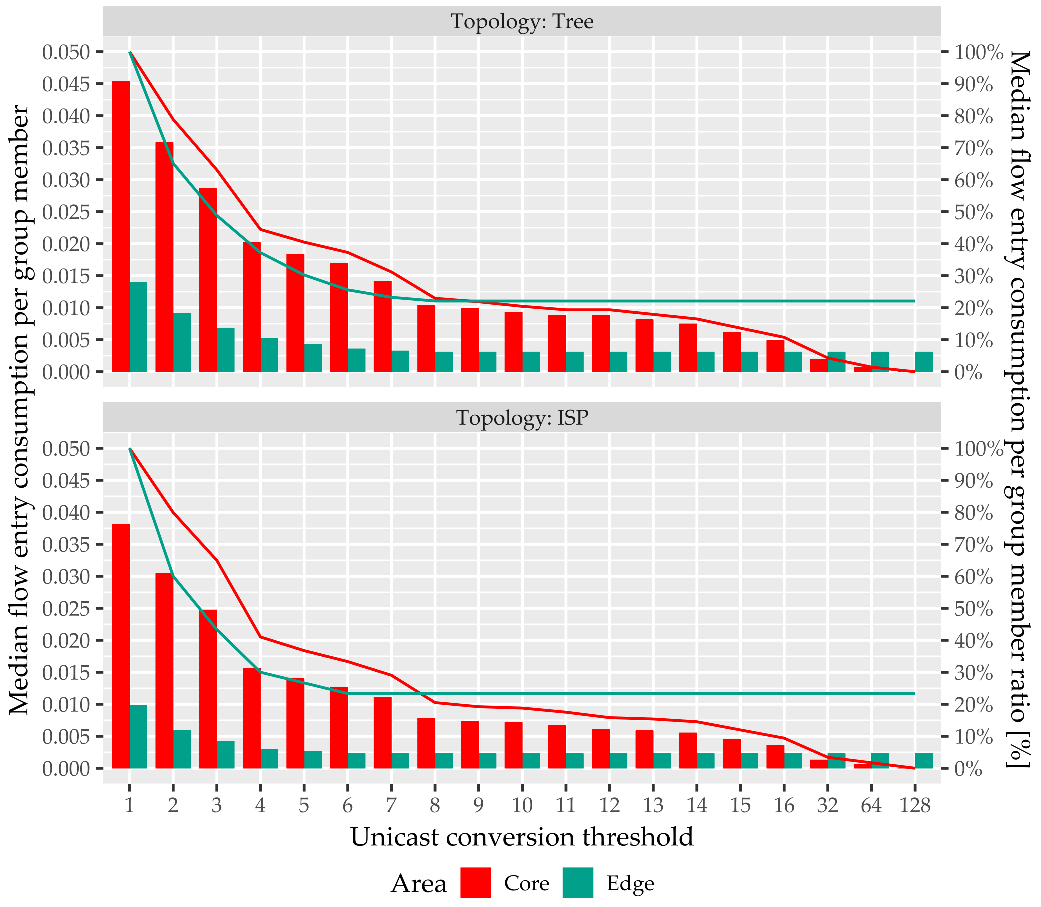


Figure 6.10: The peak state consumption on single data plane elements per network area (adapted from [Ble+15b])

relative flow entry consumption relative to its maximum. Their vertical axis is denoted on the right side of the Figure. In both, the tree, and the ISP topology the flow entry consumption per group member is higher in the core area of the network than in the edge area. This is expected because, in the tree topology, all flow entries are placed on the same devices due to the lack of alternatives.

The peak flow entries consumption in the core area is higher at most values of the unicast conversion threshold. The values where the peak consumption in the core area consumes is smaller than in the edge area are 32, 64, and 128. These values are larger than most group sizes in the investigated group size distribution. With these values, virtually all groups are immediately converted to unicast traffic. Hence, they are considered an edge case when the unicast conversion threshold is as high as or higher than the group sizes. The impact of the unicast conversion threshold on the peak flow entry consumption per group member in each area is significant for both areas up to a value of 8. After that, it stays flat for the *edge* area and continues to shrink for the *core* area. We explain this effect with the same mechanism that reduces the flow entry consumption to zero in the core area for large unicast conversion thresholds: the larger the threshold becomes in comparison to the group sizes, the more likely it becomes that each group only uses a single flow entry. This single flow entry is located on the ingress switch of the group, which is always an edge device in our design. Therefore, the flow entry consumption stays flat at the *edge* while shrinking to zero in the *core* area. We conclude that if the unicast conversion threshold is significantly smaller than the average group size, the flow entry reducing effect impacts the core and the edge area devices similarly. A flow entry shortage in the core network can be mitigated by increasing the unicast conversion threshold to a value that is close to the largest group sizes.

As we discussed, the effect of the unicast conversion threshold is different depending on the size of a multicast group. We investigate flow entry consumption per group member in more detail on the example of a single, randomly created group that grows from 1 to 128 members in Figure 6.11. When the group sizes approach values of which the unicast conversion threshold is a multiple of, spikes are visible in the state consumption. This effect is very pronounced for small group sizes and diminishes when the group becomes larger. Furthermore, the *tree* topology shows a higher variability than the *ISP* topologies. In all topologies, larger groups show less variability in their state consumption per group member. Small groups show more variability as well as a lower flow entry consumption efficiency.

For all topologies, the state efficiency increases with growing group sizes. The only exceptions in the depiction are the *ISP* topologies, here the unicast conversion threshold $T = 4$ shows an increasing trend from group size 80 to 128. We consider this a measurement artifact since all other configurations show increasing efficiency. Finally, the upper bound for the state efficiency is depicted by a unicast conversion threshold of $T = \infty$. In this case, all groups, independent of their size are converted to unicast at the group ingress switch. However, for large groups, this threshold requires a considerable replication load on a single data plane element, which we do not consider feasible.

The investigation of the data-rate-state profile showed diminishing returns for investing state to reduce the data rate and vice versa. This observation could lead to the assumption that there is an optimal unicast conversion threshold for the given network and load configuration. We investigated this finding in Figure 6.12 by assuming that both, one unit of state, and one unit of data rate, have the same value for the network operator. While this assumption is not expected to hold for many use cases, it shows that the ASDM can not only be used to adapt a multicast system to a control path resource shortage, but also to optimize the outcome of the resources invested in the system. Depending on the scarceness of state and of network capacity in the system, the optimal unicast conversion threshold is expected to change, yet the fact that is can be derived is helpful for network operators.

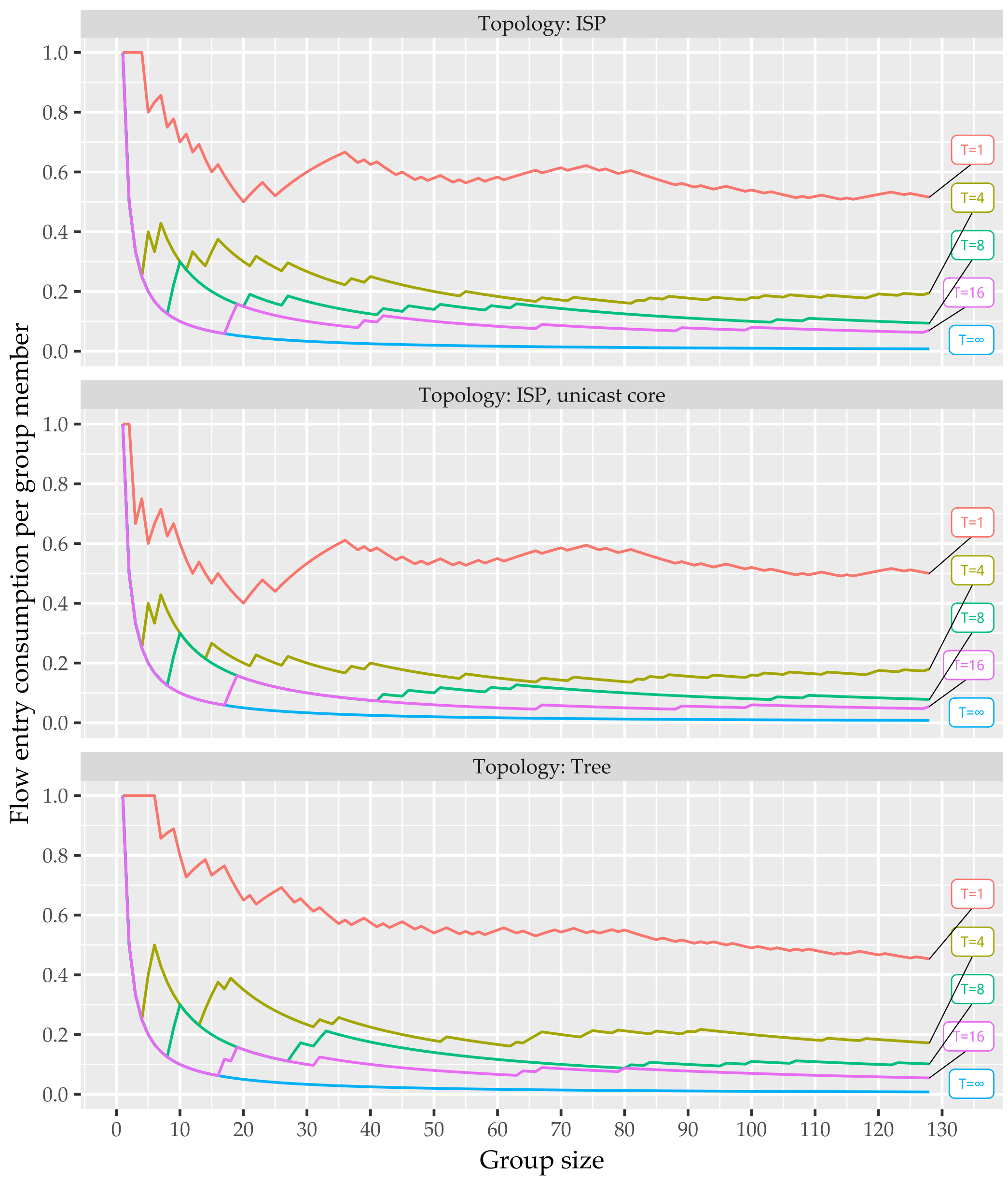


Figure 6.11: The influence of group sizes on the flow entry consumption for selected unicast conversion thresholds (adapted from [Ble+15b])

In Figure 6.12 on the horizontal axis, we see the flow entries consumed per group member, on the vertical axis the data rate reduction per flow entry. The depiction shows the data rate reduction return per invested flow entry. It is visible that the maximum return on invested flow entries is with unicast conversion threshold $T = 15$ for all three topologies. The differences between the topologies are small. We, therefore, assume that

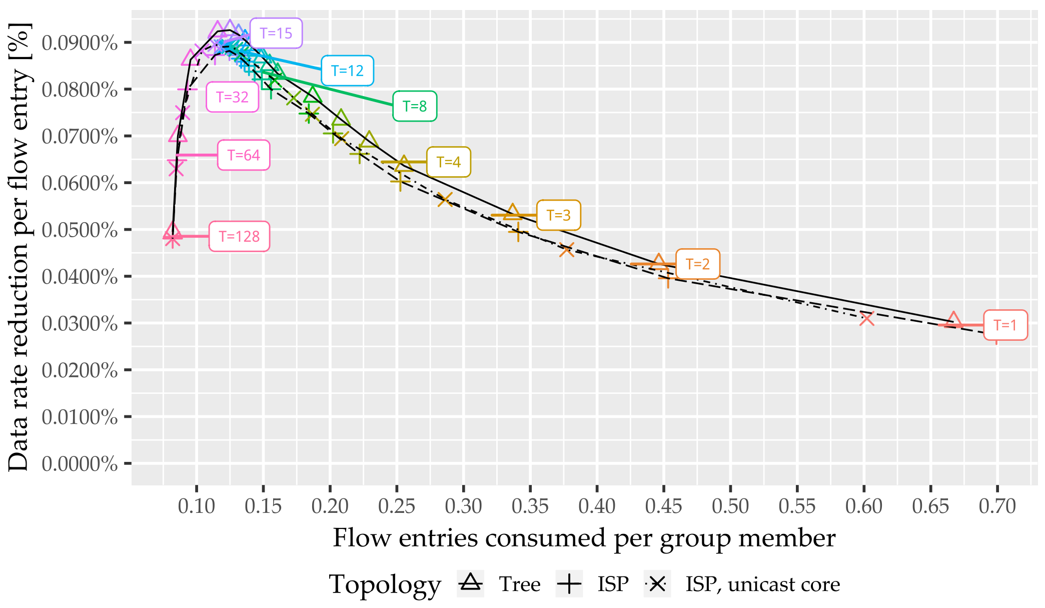


Figure 6.12: Selecting the unicast conversion threshold optimize the data rate reduction return on invested flow entries for a given cost function for data rate and state (adapted from [Ble+15b])

the optimal value depends on the valuation of data rate and state as well as the group size distribution in the ASDM control plane application.

6.2.4 *Discussion of the Evaluation Results*

We conclude this investigation of the adaptation characteristics of ASDM in three network configurations by summarizing its characteristics. We found the system to be able to reduce the state consumption by nearly 90% compared to the prevalent late duplication strategy. At the same time, the interdependent data rate efficiency can be controlled to be between ~40% and 90% of the unicast data rate.

The group size distribution is the main factor that influences the state efficiency in the system, while the transmission efficiency is also influenced by the topologies. Therefore, the decision process for selecting the unicast conversion threshold requires the system the simulated its existing group workload on the network topology to predict the outcome of a change of the threshold. To the same end, the data rate per group should be monitored with a high resolution to enable accurate predictions.

The effect of state reduction on the core and the edge areas of the ISP network showed clear differences. The edge network is always involved, while the core network can be relieved from flow entries by increasing the unicast conversion threshold to values close to the sizes of the largest groups. The tree topology showed a more pronounced behavior than the ISP topology. Specifically, due to the minimalistic nature of the topology, the state is restricted to a few data plane elements. Therefore, the peak flow entry load is higher, but so is the transmission efficiency.

An effect that was found in the evaluation but has yet to be investigated is the replication load per data plane element. For high unicast conversion thresholds, a considerable number of packet replicas might be required on a single device. The investigation if and how state-of-the-art devices can handle this load is future work.

Finally, we find that small groups are the most inefficient regarding data rates and state. We, therefore, conclude that groups that are smaller than the unicast conversion threshold should be placed on software switches. Software switches provide smaller throughput but a higher state capacity than hardware switches. Once the groups have grown large enough, they can be placed on hardware switches.

6.3 DISCUSSION

The resource-bottleneck mitigation approaches and their mitigation decision process for two representative control plane applications were evaluate in this chapter. One, network function chaining, focused on a local control path bottleneck of a renewable resource. The other, our adaptive multicasting design ASDM, investigated a global shortage of a non-renewable control path resource.

Both approaches proved to be well designed and well suited for their respective goals. The network function chaining system requires a hardware switch with a high-performance control path to be effective. The adaptive multicasting approach provides a wide control range for the data-rate-state tradeoff. Furthermore, we could show that given a valuation of data rates and state, the system can be used to select an optimal tradeoff for a given network and workload.

We found that for renewable resources separate measurements of the involved resources is not a feasible approach. Instead, the event must be tested on the actual data plane and control plane elements. The main reason for this proved to be the influence of the controller on the control path performance. Furthermore, timely performance feedback is required for this resource.

The non-renewable resource, flow entry space, promises to be an excellent candidate for using simulation for deciding which action to choose to mitigate a bottleneck. However, the adaptive multicasting approach trades flow entries for transmission capacity in the network. This is a renewable data plane resource, instead of a control plane resource. While fixed data rate traffic flows are expected for multicast traffic, a detailed measurement of the traffic flows in the multicast system is required to enable the reliable operation of the ASDM and ABSDM control plane applications.

7 SUMMARY, CONCLUSIONS, AND OUTLOOK

We conclude this thesis by summarizing it as well as giving an overview of the contributions. Finally, a conclusion is drawn, and an outlook on future work is given.

7.1 SUMMARY OF THE THESIS

The need for reliability in ISP network data planes was motivated in Chapter 1, followed by an introduction into ISP networks and SDN architectures in Chapter 2. We found that reliable control plane applications are enabled by SDN controllers that provide complete control path isolation as well as by the applications' ability to react to performance events in a controlled manner. Complete control path isolation requires SDN controllers to consider all potential performance bottlenecks on the control path and to virtualize them. Our discussion of scientific literature on potential performance bottlenecks in the SDN data plane in Chapter 3 showed that both requirements are not met by existing research. Specifically, no existing approach can identify all potential performance bottlenecks in the data plane. These findings lead us to formulate our first research goal: *Design of a systematic approach to virtualizing the control path of SDN data planes that takes all performance-relevant aspects into account*.

Furthermore, existing investigations of control plane applications do not consider the effect of control path bottlenecks on them. Therefore, the understanding of how interference from other applications in virtualized SDN data planes affect control plane applications is not sufficient. Hence, or second research goal is: *Enabling network services to operate reliably on virtualized SDN data planes*.

Based on these goals, we present our contributions to reach them in the following section.

7.2 CONTRIBUTIONS

Our answer to the first research goal and its subsequent two research questions was discussed in Chapter 6. Our first contribution is the formulation of the concept of resource-orientation and a systematic approach to control path performance analysis that provides an answer to Research Question 1.1: *How to characterize the control path performance in SDN data planes?* The performance analysis approach of the control path starts at the hardware elements, which are the cause of performance bottlenecks. We introduce an abstraction to capture the characteristics of the hardware elements, termed resources, and their interconnections through hard- and software features termed resource topology. This data structure is provided to the controller by the data plane elements and enables the controller to analytically determine performance bottlenecks in the control path before they occur.

The second Research Question 1.2 *How to virtualize the throughput aspects of control paths in SDN data planes?* was answered in the subsequent sections of Chapter 4. Using the discovered resource characteristics as well as the resource topology, the requirements for enabling an SDN controller to virtualize these resources remotely were formulated. The missing virtualization approach for throughput aspects of the control path was designed using existing mechanisms in the representatively investigated OpenFlow SDN protocol.

The systematic control path performance analysis was applied to a state-of-the-art SDN data plane element, the Edge-Core AS5712-54X switch that is based on a Broadcom Trident II packet processing ASIC. Accordingly, the throughput virtualization mechanism was evaluated. We demonstrated that the main requirements for virtualizing data planes in ISP networks, prioritization could be achieved by our design while the fairness was increased. Moreover, the presented approach is faster and more efficient than the prevalent method used today.

The solution for the second research goal is presented in Chapters 5 and 6. First, the types of resource events in control paths of SDN data planes are discussed. Then, a design space analysis of bottleneck mitigation approaches is conducted.

We investigated the effects of control path resource bottlenecks and the corresponding mitigation approaches on two representative applications, network function chaining and SDM that cover all relevant parts of an ISP network. Research Question 2.1 *How can control plane applications operate reliably in the face of control path performance events that affect a single data plane element?* is investigated by enabling a network function chaining control plane application to cope with performance bottlenecks in the control path. To that end, we analyze the requirements for shifting the load of flow table entry additions from an SDN software switch to an adjacent hardware switch in our network function chaining application. Research Question 2.2 *How can control plane applications operate reliably in the face of control path performance events that affect the whole data plane of a network domain?* is investigated by designing an adaptive SDM control plane application, termed ASDM, from the beginning for the ability to adapt to changing control path resource availability. Furthermore, the adaptivity concept is applied to the efficient bit-index replication forwarding method, to further increase the adaptivity and efficiency of multicasting in our ABSDM application design. The designs are evaluated in Chapter 6, and both the effectiveness of the mitigation approaches is demonstrated, as well as the application's requirements for determining their mitigation decision. We find that both applications can reliably react to the investigated cases of control path resource bottlenecks and thereby ensure the reliability of the ISP network control plane operating on a virtualized data planed.

7.3 CONCLUSION

We systematically analyzed the control path of state-of-the-art SDN hardware switches to enable its virtualization. Furthermore, we studied the impact of virtualization on representative control plane applications.

SDN protocols need to take the control path of the data plane into account and to be able to reason on its expected performance as well as its current state. Control plane applications need to be able to specify and receive information on their used control path resources. This information enables applications to operate reliably in the face of performance events.

The investigation of a state-of-the-art OpenFlow agent for hardware switches, PicOS 2.8 revealed that the design of operating systems for the management system of hardware switches has not caught up with the research yet. Specifically, the design of existing OpenFlow agents hinders the reasoning on the performance of individual SDN primitives.

We, therefore, conclude that performance clarity should be a first-class design principle for both hardware and software designs as proposed by Ousterhout et at [Ous+17]. Thereby, making SDN control planes in future ISP networks more efficient and reliable.

7.4 OUTLOOK

Our approach to deriving a resource topology was successfully applied to a state-of-the-art SDN switch. The design focused on devices relying on ASICs for packet processing. However, there are other classes of data plane elements, such as software switches as well as NPU-based and FPGA-based devices. Software and NPU-based data plane elements rely to a more considerable extent on software to implement packet processing. Therefore, the mapping of SDN protocol primitives to hardware resources becomes more difficult and requires resource isolation between software features. The impact of the increased importance of software components on our resource-oriented performance analysis approach as well as the requirements for software to provide performance clarity should be investigated.

The requirements for providing more performance clarity to software in data planes also applies to the SDN agent software running on the device's management system. Today, all SDN primitives are processed by the same software components consuming the same resources. Ensuring that important SDN protocol messages are still processed when the management system is overloaded is difficult. Therefore, an approach to improve the reliability of message delivery should be investigated as well as ways to provide isolation between different priorities of SDN protocol messages. Doing so requires vendors to open their hardware drivers and make them available without the requirement for an NDA. Nevertheless, the proposed system is well suited as the foundation of an experimental SDN agent for data plane elements to demonstrate all features at once.

Finally, the resource-orientation provides new possibilities for network management. Using our contributions, control plane applications can be compared based on their data plane resource consumption. Moreover, the efficiency of data plane elements in hosting control plane applications becomes comparable by matching the resource requirements of control plane applications with the available resources in the data plane. With this thesis at hand, we provide the foundation to enable network management software to automatically select the best-suited control plane applications for an existing network.

7.5 FUNDING

This work has been supported in parts by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 - MAKI, by the project „Dynamic Networks (D-Nets) 2-6“ in collaboration with Deutsche Telekom FMED, and by the European Union (FP7/#317846, SmartenIT and FP7/#318398, eCousin).

BIBLIOGRAPHY

[AHA16] G. Abbas, Z. Halim, and Z. H. Abbas. „Fairness-Driven Queue Management: A Survey and Taxonomy.“ In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 324–367.

[AH02] L. A. Adamic and B. A. Huberman. „Zipf's law and the Internet.“ In: *Glottometrics* 3.1 (2002), pp. 143–150.

[Aga+14] K. Agarwal, C. Dixon, E. Rozner, and J. Carter. „Shadow MACs: Scalable Label-switching for Commodity Ethernet.“ In: *Proceedings of the Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 2014.

[Amb+17] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran. „LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking.“ In: *IEEE/ACM Transactions on Networking* 25.2 (Apr. 2017), pp. 1206–1219.

[AC05] G. Apostolopoulos and I. Ciurea. „Reducing the forwarding state requirements of point-to-multipoint trees using MPLS multicast.“ In: *Proceedings of the International Symposium on Computers and Communications (ISCC)*. 2005.

[Bas+17] A. Basta, A. Blenk, W. Kellerer, and S. Schmid. „Logically Isolated, Actually Unpredictable? Measuring Hypervisor Performance in Multi-Tenant SDNs.“ In: *Computing Research Repository (CoRR)* abs/1704.08958 (2017). arXiv: 1704.08958.

[Ber+14] P. Berde et al. „ONOS: Towards an Open, Distributed SDN OS.“ In: *Proceedings of the Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 2014.

[Bet+14] A. Betker, I. Gamrath, D. Kosiankowski, C. Lange, H. Lehmann, F. Pfeuffer, F. Simon, and A. Werner. „Comprehensive Topology and Traffic Model of a Nationwide Telecommunication Network.“ In: *Journal of Optical Communications and Networking* 6.11 (Nov. 2014), pp. 1038–1047.

[Bho+17] S. Bhowmik, M. A. Tariq, B. Koldehofe, F. Durr, T. Kohler, and K. Rothermel. „High Performance Publish/Subscribe Middleware in Software-Defined Networks.“ In: *IEEE/ACM Transactions on Networking* 25.3 (June 2017), pp. 1501–1516.

[Bif+13] R. Bifulco, T. Dietz, F. Huici, M. Ahmed, J. Martins, S. Niccolini, and H. Kolbe. „Rethinking Access Networks with High Performance Virtual Software BRASes." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2013.

[Ble15] S. Bleidner. „Identification and Design of a Data Plane Resource Optimization Mechanism for Application-Controlled SDN." PS-D-0017. Master's thesis. Technische Universität Darmstadt, 2015.

[Ble+16a] J. Blendin, Y. Babenko, D. Kusidlo, G. Schyguda, and D. Hausheer. „Position Paper: Towards a Structured Approach to Developing Benchmarks for Virtual Network Functions." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2016.

[BH14] J. Blendin and D. Hausheer. „Towards Resource-Efficient Application-Controlled Software Defined Networks." In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. Phd paper. 2014.

[Ble+16b] J. Blendin, D. Herrmann, M. Wichtlhuber, M. Gunkel, F. Wissel, and D. Hausheer. „Enabling efficient multi-layer repair in elastic optical networks by gradually superimposing SDN." In: *Proceedings of the International Conference on Network and Service Management (CNSM)*. 2016.

[Ble+15a] J. Blendin, J. Rückert, S. Bleidner, and D. Hausheer. „Taking the Sting out of Flow Update Peaks in Software-Defined Service Chaining." In: *Proceedings of the International Workshop on Management of SDN and NFV Systems (ManSDN/NFV)*. 2015.

[Ble+14] J. Blendin, J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer. „Position Paper: Software-Defined Network Service Chaining." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2014.

[Ble+15b] J. Blendin, J. Rückert, T. Volk, and D. Hausheer. „Adaptive Software Defined Multicast." In: *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*. 2015.

[Ble13] J. Blendin. „Cross-layer Optimization of Peer-to-Peer Video Streaming in OpenFlow-based ISP Networks." Diploma Thesis. Technische Universität Darmstadt, 2013.

[Ble+18] J. Blendin, F. Bendfeldt, I. Poese, B. Koldehofe, and O. Hohlfeld. „Dissecting Apple's Meta-CDN during an iOS Update." In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. Accepted. 2018.

[BBK15] A. Blenk, A. Basta, and W. Kellerer. „HyperFlex: An SDN Virtualization Architecture with Flexible Hypervisor Function Allocation." In: *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2015.

[Ble+15c] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer. „Survey on Network Virtualization Hypervisors for Software Defined Networking.“ In: *Computing Research Repository (CoRR)* abs/1506.07275 (2015). arXiv: 1506.07275.

[Ble+16c] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer. „Survey on Network Virtualization Hypervisors for Software Defined Networking.“ In: *IEEE Communications Surveys Tutorials* 18.1 (Firstquarter 2016), pp. 655–685.

[RFC5058] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms. *Explicit Multicast (Xcast) Concepts and Options*. Request for Comments 5058. Internet Engineering Task Force, Nov. 2007.

[Bos+13] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. „Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2013.

[BC02] A. Boudani and B. Cousin. „A new approach to construct multicast trees in MPLS networks.“ In: *Proceedings of the International Symposium on Computers and Communications (ISCC)*. 2002.

[BR13] Z. Bozakov and A. Rizk. „Taming SDN Controllers in Heterogeneous Hardware Environments.“ In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2013.

[BS15] E. Bulut and B. K. Szymanski. „Understanding user behavior via mobile data analysis.“ In: *Proceedings of the IEEE International Conference on Communication Workshop (ICCW)*. 2015.

[CR17] R. Canonico and S. P. Romano. „Leveraging SDN to Improve the Performance of Multicast-Enabled IPTV Distribution Systems.“ In: *IEEE Communications Standards Magazine* 1.4 (Dec. 2017), pp. 42–47.

[CTB16] B. Chandrasekaran, B. Tschaen, and T. Benson. „Isolating and Tolerating SDN Application Failures with LegoSDN.“ In: *Proceedings of the Symposium on SDN Research (SOSR)*. 2016.

[CB17a] H. Chen and T. Benson. „Hermes: Providing Tight Control over High-Performance SDN Switches.“ In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2017.

[CB17b] H. Chen and T. Benson. „The Case for Making Tight Control Plane Latency Guarantees in SDN Switches.“ In: *Proceedings of the Symposium on SDN Research (SOSR)*. 2017.

[CVNI17] *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021*. Whitepaper. Cisco. June 2017.

[Cos+17] L. C. Costa, A. B. Vieira, E. d. B. e. Silva, D. F. Macedo, G. Gomes, L. H. A. Correia, and L. F. M. Vieira. „Performance evaluation of OpenFlow data planes." In: *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2017.

[Csá+13] A. Császár, W. John, M. Kind, C. Meirosu, G. Pongrácz, D. Staessens, A. Takács, and F.-J. Westphal. „Unifying Cloud and Carrier Network: EU FP7 Project UNIFY." In: *Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. 2013.

[Cur+11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. „DevoFlow: Scaling Flow Management for High-performance Networks." In: *ACM SIGCOMM Computer Communication Review (CCR)* 41.4 (2011), pp. 254–265.

[DPM12] S. Das, G. Parulkar, and N. McKeown. „Why OpenFlow/SDN can Succeed where GMPLS Failed." In: *Proceedings of the European Conference and Exhibition on Optical Communication (ECOC)*. 2012.

[DC90] S. E. Deering and D. R. Cheriton. „Multicast Routing in Datagram Internetworks and Extended LANs." In: *ACM Transactions on Computer Systems (TOCS)* 8.2 (May 1990), pp. 85–110.

[Dio+00] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefen. „Deployment Issues for the IP Multicast Service and Architecture." In: *IEEE Network* 14.1 (2000), pp. 78–88.

[DKE14] A. Dixit, K. Kogan, and P. Eugster. „Composing Heterogeneous SDN Controllers with Flowbricks." In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. 2014.

[RFC5810] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. Request for Comments 5810. Internet Engineering Task Force, Mar. 2010.

[DRC10] R. D. Doverspike, K. K. Ramakrishnan, and C. Chase. „Structural Overview of ISP Networks." In: *Guide to Reliable Internet Services and Applications*. Ed. by C. R. Kalmanek, S. Misra, and Y. Yang. Springer London, 2010, pp. 19–93. ISBN: 978-1-84882-828-5.

[DK15] R. Durner and W. Kellerer. „The cost of security in the SDN control plane." In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT) Student Workshop*. 2015.

[DG06] M. Düser and A. Gladisch. „Evaluation of Next Generation Network Architectures and Further Steps for a Clean Slate Networking Approach." In: *Proceedings of the Joint ITG and Euro-NF Workshop "Visions of Future Generation Networks" (EuroView)*. 2006.

[Emm+15] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle. „Assessing Soft- and Hardware Bottlenecks in PC-based Packet Forwarding Systems.“ In: *Proceedings of the International Conference on Networks (ICN)*. 2015.

[Far10] A. Farrel. „A Unified Control Plane: Dream or Pipedream.“ In: *Proceedings of the International Conference on IP+ Optical Network (IPOP)*. 2010.

[Fil+18] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li. *SRv6 Network Programming*. Internet-Draft draft-filsfils-spring-srv6-network-programming-05. Internet Engineering Task Force, July 2018.

[FSV16] K. Foerster, S. Schmid, and S. Vissicchio. „Survey of Consistent Network Updates.“ In: *Computing Research Repository (CoRR)* abs/1609.02305 (2016). arXiv: 1609.02305.

[Gio+17] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini. „First demonstration of SDN-based Bit Index Explicit Replication (BIER) multicasting.“ In: *Proceedings of the European Conference on Networks and Communications (EuCNC)*. 2017.

[Gre13] B. Gregg. „Thinking Methodically About Performance.“ In: *Communications of the ACM* 56.2 (Feb. 2013), pp. 45–51.

[Guo+17] Z. Guo, R. Liu, Y. Xu, A. Gushchin, A. Walid, and H. J. Chao. „STAR: Preventing flow-table overflow in software-defined networks.“ In: *Computer Networks* 125 (2017), pp. 15–25.

[Här17] M. Härdtlein. „Identification and Modeling Performance Interference in Virtual SDN Data Planes.“ KOM-B-0602. Bachelor's thesis. Technische Universität Darmstadt, 2017.

[Har+16] R. Hark, D. Stingl, N. Richerzhagen, K. Nahrstedt, and R. Steinmetz. „DistTM: Collaborative traffic matrix estimation in distributed SDN control planes.“ In: *Proceedings of the IFIP Networking Conference*. 2016.

[He+15] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan. „Measuring Control Plane Latency in SDN-enabled Switches.“ In: *Proceedings of the Symposium on SDN Research (SOSR)*. 2015.

[HYS] D. Y. Huang, K. Yocum, and A. C. Snoeren. „High-fidelity Switch Models for Software-defined Network Emulation.“ In: *Proceedings of the Workshop on Hot Topics in Software Defined Networking (HotSDN)*.

[IMS13] A. S. Iyer, V. Mann, and N. R. Samineni. „SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks.“ In: *Proceedings of the IFIP Networking Conference*. 2013.

[Jai+13] S. Jain et al. „B4: Experience with a Globally-deployed Software Defined Wan.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2013.

[Jar+11] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia. „Modeling and Performance Evaluation of an OpenFlow Architecture." In: *Proceedings of the International Teletraffic Congress (ITC)*. 2011.

[Jin+15] X. Jin, J. Gossels, J. Rexford, and D. Walker. „CoVisor: A Compositional Hypervisor for Software-defined Networks." In: *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 2015.

[Jin+13] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. „SoftCell: Scalable and Flexible Cellular Core Network Architecture." In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2013.

[Jin+14] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. „Dynamic Scheduling of Network Updates." In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2014.

[KHK13] Y. Kanizo, D. Hay, and I. Keslassy. „Palette: Distributing tables in software-defined networks." In: *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 2013.

[Kat+16] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. „CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks." In: *Proceedings of the Symposium on SDN Research (SOSR)*. 2016.

[KMH14] F. Kaup, S. Melnikowitsch, and D. Hausheer. „Measuring and modeling the power consumption of OpenFlow switches." In: *Proceedings of the International Conference on Network and Service Management (CNSM)*. 2014.

[Kni+11] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. „The Internet Topology Zoo." In: *IEEE Journal on Selected Areas in Communications* 29.9 (Oct. 2011), pp. 1765–1775.

[Koh+07] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. „An Analysis of Performance Interference Effects in Virtual Environments." In: *Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. 2007.

[Kop+10] T. Koponen et al. „Onix: A Distributed Control Platform for Large-scale Production Networks." In: *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*. 2010.

[Kre+15] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. „Software-Defined Networking: A Comprehensive Survey." In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

[RFC1736] J. Kunze. *Functional Recommendations for Internet Resource Locators*. Request for Comments 1736. Internet Engineering Task Force, Feb. 1995.

[Kuz+18] M. Kuzniar, P. Peresini, D. Kostic, and M. Canini. „Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches." In: *Computer Networks* 136 (May 2018), pp. 22–36.

[LHM10] B. Lantz, B. Heller, and N. McKeown. „A Network in a Laptop: Rapid Prototyping for Software-defined Networks." In: *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. 2010.

[Laz+14] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. „Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization." In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2014.

[Lev+14] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann. „Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks." In: *Proceeding of the USENIX Annual Technical Conference (USENIX ATC)*. 2014.

[Liu+12] L. Liu et al. „First Field Trial of an OpenFlow-based Unified Control Plane for Multi-layer Multi-granularity Optical Networks." In: *Proceedings of the Optical Fiber Communication Conference (OFC)*. 2012.

[Mad+15] A. Madhavapeddy et al. „Jitsu: Just-In-Time Summoning of Unikernels." In: *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 2015.

[Mai+09] G. Maier, A. Feldmann, V. Paxson, and M. Allman. „On Dominant Characteristics of Residential Broadband Internet Traffic." In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2009.

[Man+17] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici. „My VM is Lighter (and Safer) Than Your Container." In: *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. 2017.

[RFC3945] E. Mannie. *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*. Request for Comments 3945. Internet Engineering Task Force, Oct. 2004.

[McK03] N. McKeown. *Processing packets in packet switches*. Stanford University, Lecture CS343. May 2003.

[McK09] N. McKeown. *Software-defined networking*. IEEE International Conference on Computer Communications (INFOCOM) keynote talk. Apr. 2009.

[Med+14] J. Medved, R. Varga, A. Tkacik, and K. Gray. „OpenDaylight: Towards a Model-Driven SDN Controller architecture." In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 2014.

[MM16] T. Mizrahi and Y. Moses. „Time4: Time for SDN.“ In: *IEEE Transactions on Network and Service Management (TNSM)* 13.3 (2016), pp. 433–446.

[Mog+13] J. C. Mogul, A. AuYoung, S. Banerjee, L. Popa, J. Lee, J. Mudigonda, P. Sharma, and Y. Turner. „Corybantic: Towards the Modular Composition of SDN Control Programs.“ In: *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. 2013.

[Mos+14] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. „DREAM: Dynamic Resource Allocation for Software-defined Measurement.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2014.

[Nar+12] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam. „Macroflows and Microflows: Enabling Rapid Network Innovation through a Split SDN Data Plane.“ In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2012.

[ETSI13] *Network Functions Virtualisation (NFV); Architectural Framework*. Tech. rep. ETSI GS NFV 002 V1.1.1. ETSI Industry Specification Group (ISG), Network Functions Virtualisation (NFV), Oct. 2013.

[ETSI18] *Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV*. Tech. rep. ETSI GS NFV 003 V1.3.1. ETSI Industry Specification Group (ISG), Network Functions Virtualisation (NFV), Jan. 2018.

[Ngu+18] A. Nguyen-Ngoc, S. Lange, S. Geissler, T. Zinner, and P. Tran-Gia. „Estimating the Flow Rule Installation Time of SDN Switches When Facing Control Plane Delay.“ In: *Proceedings of the International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*. 2018.

[Nir+09] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. „PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2009.

[Nob+17] L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, and D. Hausheer. „Bare-Metal Switches and Their Customization and Usability in a Carrier-Grade Environment.“ In: *Proceeding of the IEEE Conference on Local Computer Networks (LCN)*. 2017.

[NHH16] L. Nobach, O. Hohlfeld, and D. Hausheer. „New Kid on the Block: Network Functions Visualization: From Big Boxes to Carrier Clouds.“ In: *ACM SIGCOMM Computer Communication Review (CCR)* 46.3 (July 2016), 7:1–7:8.

[ONF14a] *OF-CONFIG 1.2, OpenFlow Management and Configuration Protocol*. Technical Specification 016. Open Networking Foundation, 2014.

[ONF09] *OpenFlow Switch Specification: Version 1.0.0 (Wire Protocol 0x01)*. Technical Specification 001. Open Networking Foundation, Dec. 2009.

[ONF13] *OpenFlow Switch Specification: Version 1.4.0 (Wire Protocol 0x05)*. Technical Specification 012. Open Networking Foundation, Oct. 2013.

[ONF15] *OpenFlow Switch Specification: Version 1.5.1 (Protocol version 0x06)*. Technical Specification 025. Open Networking Foundation, Mar. 2015.

[ONF14b] *OpenFlow Table Type Patterns*. Technical Specification 017. Open Networking Foundation, Aug. 2014.

[ONF17] *Orchestration: A More Holistic View*. Technical Reference 540. Open Networking Foundation, Mar. 2017.

[Ous+17] K. Ousterhout, C. Canel, M. Wolffe, S. Ratnasamy, and S. Shenker. „Performance Clarity As a First-class Design Principle." In: *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*. 2017.

[P4Runtime] *P4Runtime Specification*. Tech. rep. version 1.0.0-rc2. The P4.org API Working Group, Mar. 2018.

[PS06] K. Pagiamtzis and A. Sheikholeslami. „Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey." In: *IEEE Journal of Solid-State Circuits* 41.3 (2006), pp. 712–727.

[PB17] D. Pariag and T. Brecht. „Application Bandwidth and Flow Rates from 3 Trillion Flows Across 45 Carrier Networks." In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2017.

[PV11] D. Perino and M. Varvello. „A Reality Check for Content Centric Networking." In: *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*. 2011.

[Pet+16] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow. „Central office re-architected as a data center." In: *IEEE Communications Magazine* 54.10 (Oct. 2016), pp. 96–101.

[RFC7047] B. Pfaff and B. Davie. *The Open vSwitch Database Management Protocol*. Request for Comments 7047. Internet Engineering Task Force, Dec. 2013.

[Pfa+15] B. Pfaff et al. „The Design and Implementation of Open vSwitch." In: *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. 2015.

[PRF17] E. Pujol, P. Richter, and A. Feldmann. „Understanding the Share of IPv6 Traffic in a Dual-Stack ISP." In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2017.

[Qaz+13] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. „SIMPLE-fying Middlebox Policy Enforcement Using SDN.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2013.

[Qia+16] S. Qiao, C. Hu, X. Guan, and J. Zou. „Taming the Flow Table Overflow in OpenFlow Switch.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2016.

[RFC8300] P. Quinn, U. Elzur, and C. Pignataro. *Network Service Header (NSH)*. Request for Comments 8300. Internet Engineering Task Force, Jan. 2018.

[RLANG] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL: `https://www.R-project.org/`.

[REG99] P. I. Radoslavov, D. Estrin, and R. Govindan. *Exploiting the bandwidth-memory tradeoff in multicast state aggregation*. Technical Report 99-697 (Second Revision). University of Southern California, Department of Computer Science, 1999.

[Roj+18] E. Rojas, R. Doriguzzi-Corin, S. Tamurejo, A. Beato, A. Schwabe, K. Phemius, and C. Guerrero. „Are We Ready to Drive Software-Defined Networks? A Comprehensive Survey on Management Tools and Techniques.“ In: *ACM Computing Surveys (CSUR)* 51.2 (Feb. 2018), 27:1–27:35.

[Rot+12] C. Rotsos, N. Sarrar, S. Uhlig, and A. W. Sherwood Rob and Moore. „OFLOPS: An Open Framework for OpenFlow Switch Evaluation.“ In: *Proceedings of the International Conference on Passive and Active Network Measurement (PAM)*. 2012.

[Rüc+15] J. Rückert, J. Blendin, R. Hark, and D. Hausheer. „DynSDM: Dynamic and Flexible Software-Defined Multicast for ISP Environments.“ In: *Proceedings of the International Conference on Network and Service Management (CNSM)*. 2015.

[Rüc+16] J. Rückert, J. Blendin, R. Hark, and D. Hausheer. „Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSDM.“ In: *IEEE Transactions on Network and Service Management (TNSM)* 13.4 (2016), pp. 754–767.

[RBH13] J. Rückert, J. Blendin, and D. Hausheer. „RASP: Using OpenFlow to Push Overlay Streams into the Underlay (Demo Paper).“ In: *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*. 2013.

[RBH15] J. Rückert, J. Blendin, and D. Hausheer. „Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks.“ In: *Springer Journal of Network and Systems Management (JNSM), Special Issue on Management of Software-Defined Networks* 23.2 (2015), pp. 280–308.

[Rüc+14] J. Rückert, J. Blendin, N. Leymann, G. Schyguda, and D. Hausheer. „Demo: Software-Defined Network Service Chaining." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2014.

[Rüc16] J. Rückert. *Large-Scale Live Video Streaming Over the Internet*. Verlag Dr. Hut, 2016. ISBN: 978-3-8439-2836-6.

[RFC3549] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. *Linux Netlink as an IP Services Protocol*. Request for Comments 3549. Internet Engineering Task Force, July 2003.

[SPA16] T. Sasaki, A. Perrig, and D. E. Asoni. „Control-plane isolation and recovery for a secure SDN architecture." In: *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*. 2016.

[SAK16] A. Schwabe, P. A. Aranda Gutiérrez, and H. Karl. „Composition of SDN Applications: Options/Challenges for Real Implementations." In: *Proceedings of the Applied Networking Research Workshop (ANRW)*. 2016.

[SK14] A. Schwabe and H. Karl. „Using MAC Addresses As Efficient Routing Labels in Data Centers." In: *Proceedings of the Workshop on Hot Topics in Software Defined Networking (HotSDN)*. 2014.

[SNS16] S. Scott-Hayward, S. Natarajan, and S. Sezer. „A Survey of Security in Software Defined Networks." In: *IEEE Communications Surveys Tutorials* 18.1 (2016), pp. 623–654.

[ONF14c] *SDN architecture*. Technical Reference 502. Open Networking Foundation, June 2014.

[She+09] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. „Flowvisor: A Network Virtualization Layer." In: *OpenFlow Switch Consortium, Technical Report* (2009).

[She+10] R. Sherwood et al. „Carving Research Slices out of Your Production Networks with OpenFlow." In: *ACM SIGCOMM Computer Communication Review (CCR)* 40.1 (Jan. 2010), pp. 129–130.

[Shi+14] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. „Rosemary: A Robust, Secure, and High-performance Network Operating System." In: *Proceedings of the ACM Conference on Computer and Communications Security (CSS)*. 2014.

[SJ13] P. Sköldström and W. John. „Implementation and Evaluation of a Carrier-Grade OpenFlow Virtualization Scheme." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2013.

[SY12] P. Sköldström and K. Yedavalli. „Network virtualization and resource allocation in OpenFlow-based wide area networks." In: *Proceedings of the IEEE International Conference on Communications (ICC)*. 2012.

[Sou+14] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster. „Merlin: A Language for Provisioning Network Resources.“ In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2014.

[SMZ04] K. Sripanidkulchai, B. Maggs, and H. Zhang. „An Analysis of Live Streaming Workloads on the Internet.“ In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. 2004.

[SNZ00] I. Stoica, T. Ng, and H. Zhang. „REUNITE: a Recursive Unicast Approach to Multicast.“ In: *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. 2000.

[P4_14] *The P4 Language Specification*. Tech. rep. Version 1.0.4. The P4 Language Consortium, May 2017.

[TL01] S.-C. Tsao and Y.-D. Lin. „Pre-order deficit round robin: a new scheduling algorithm for packet-switched networks.“ In: *Computer Networks* 35.2 (2001), pp. 287–305.

[Vil18] L. F. Villa-Arenas. „Ensuring Fairness and Resilience for Flow Rule Updates in Virtualized SDNs.“ KOM-M-0608. Master’s thesis. Technische Universität Darmstadt, 2018.

[Vol14] T. Volk. „Supporting Multicast in Application-controlled Software Defined Networks.“ PS-D-0012. Master’s thesis. Technische Universität Darmstadt, 2014.

[Wan+14] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. „Scotch: Elastically Scaling Up SDN Control-Plane Using vSwitch Based Overlay.“ In: *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 2014.

[Wel16] P. Welzel. „Bit-Indexed Software Defined Multicast.“ PS-D-0033. Master’s thesis. Technische Universität Darmstadt, 2016.

[Wen+16a] X. Wen, B. Yang, Y. Chen, L. E. Li, K. Bu, P. Zheng, Y. Yang, and C. Hu. „RuleTris: Minimizing Rule Update Latency for TCAM-Based SDN Switches.“ In: *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. 2016.

[Wen+16b] X. Wen, B. Yang, Y. Chen, C. Hu, Y. Wang, B. Liu, and X. Chen. „SDNShield: Reconciliating Configurable Application Permissions for SDN App Markets.“ In: *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2016.

[Wet+14] P. Wette, M. Dräxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl. „MaxiNet: Distributed emulation of software-defined networks.“ In: *Proceedings of the IFIP Networking Conference*. 2014.

[RFC8279] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin. *Multicast Using Bit Index Explicit Replication (BIER)*. Request for Comments 8279. Internet Engineering Task Force, Nov. 2017.

[Wil+97] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. „Self-similarity Through High-variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level.“ In: *IEEE/ACM Transactions on Networking* 5.1 (Feb. 1997), pp. 71–86.

[Yoo+] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu. „Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks.“ In: *IEEE/ACM Transactions on Networking* 25.6 (), pp. 3514–3530.

[Yu+10] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. „Scalable Flow-based Networking with DIFANE.“ In: *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*. 2010.

[Zha18] X. Zhang. „Characterizing the Packet Duplication Behavior of State-of-the-Art SDN Switches.“ KOM-M-0610. Master's thesis. Technische Universität Darmstadt, 2018.

[Zha+13] Y. Zhang et al. „StEERING: A software-defined networking for inline service chaining.“ In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. 2013.

[Zin+14] T. Zinner, M. Jarschel, A. Blenk, F. Wamser, and W. Kellerer. „Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges.“ In: *Proceeding of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2014.

All web pages cited in this work have been checked in September 2018. However, because of the dynamic nature of the World Wide Web, their long-term availability cannot be guaranteed.

ACRONYMS

ABSDM	Adaptive Bit-Index Software-Defined Multicast
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
ASDM	Adaptive Software-Defined Multicast
ASIC	Application-specific Integrated Circuit
BCAM	Binary Content-addressable Memory
BGP	Border Gateway Protocol
BIER	Bit Indexed Explicit Replication
CAM	Content-addressable Memory
CDN	Content Delivery Network
CLI	Command Line Interface
CORD	Central Office Re-architected As A Data Center
COTS	Commercial Off-the-shelf
CPU	Central Processing Unit
ECMP	Equal-cost Multi-path
ETSI	European Telecommunications Standards Institute
FIFO	First In, First Out
ForCES	Forwarding And Control Element Separation
FPGA	Field-programmable Gate Array
GMPLS	Generalized Multi-Protocol Label Switching
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPTV	Internet Protocol Television
ISP	Internet Service Provider
MPLS	Multi-Protocol Label Switching
NDA	Non-disclosure Agreement
NFV	Network Functions Virtualization

NIC	Network Interface Card
NPU	Network Processing Unit
OF-DPA	OpenFlow - Data Plane Abstraction
TTP	Table Type Pattern
ONF	Open Networking Foundation
OS	Operating System
OSPF	Open Shortest Path First
OTT	Over-the-top
PCI	Peripheral Component Interconnect
PCIe	PCI Express
ROADM	Reconfigurable Optical Add-drop Multiplexer
SCTP	Stream Control Transmission Protocol
SDM	Software-Defined Multicast
SDN	Software-defined Networking
SoC	System On Chip
TCP	Transmission Control Protocol
ToR	Top-of-rack
UDP	User Datagram Protocol
USE	Utilization, Saturation, Errors
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function
VoIP	Voice Over IP

GLOSSARY

Arista A network equipment vendor. 53

Barefoot Tofino A fully programmable data plane chip. 53
barrier An OpenFlow primitive. 72, 117, 118
barrier_reply An OpenFlow message type. 57, 72, 117
barrier_request An OpenFlow message type. 72
Broadcom A switch ASIC vendor. 29, 30, 51–53, 63, 64, 66, 67, 76, 138, 161
bundle_commit_reply An OpenFlow primitive. 72, 73

control plane application A SDN control plane application. iii, 1–7, 17, 19, 21, 26, 27, 30, 33, 35, 37–40, 43–45, 49–51, 58, 65, 67–69, 72, 73, 75, 77, 78, 81–84, 86–88, 98–100, 102, 111–113, 115, 121, 124, 125, 129, 134, 135, 137–140

Dell 8132F A Broadcom Trident+-based OpenFlow-enabled switch. 75, 76
Delta Networks A network equipment vendor. 53

Edge-Core AS5712-54X A Broadcom Trident II-based bare-metal switch. ix–xi, 5, 51–55, 57, 59, 64–68, 74–77, 82, 117, 125, 138, 161–163, 165, 167–169
Ethernet The dominant ISO/OSI layer two protocol used today. 62

Floodlight A state-of-the-art OpenFlow controller. 73, 75
flow bundle An OpenFlow primitive. 72–80, 82
flow_mod An OpenFlow message type. 24, 65, 68, 69, 71, 73–75, 162

GbE Gigabit Ethernet 5, 75, 117
global resource event A resource event that affects a whole area of the data plane of a network domain. 4, 84
group_mod An OpenFlow message type. 65, 66

HP A network equipment vendor. 29, 52–54

Linux An open-source operating system. 30, 52, 58
local resource event A resource event that affects a single data plane element of a network domain. 4, 84, 86

management system An embedded computer that manages the switch ASIC and provides an interface to the control plane. 11, 12, 16, 20, 21, 24–27, 29, 30, 42, 46, 50–55, 57, 58, 62, 64–67, 75, 117, 139, 161, 169
meter_mod An OpenFlow message type. 65, 66

NEC A network equipment vendor. 29, 52, 53
NoviFlow A network equipment vendor. 29, 52

OF-CONFIG The OpenFlow Management and Configuration Protocol. 23, 43, 49, 59, 65
Open vSwitch An OpenFlow software switch. 39, 51, 53–56, 120–122, 124, 128
OpenFlow An SDN protocol defined by the ONF. x, 14, 16, 21, 23–30, 32, 39, 40, 42–44, 46, 47, 49–75, 79, 80, 82, 88, 93, 94, 96–98, 100, 108–111, 116–118, 120, 128, 138, 139, 161, 162, 165
ovs-vswitchd The OpenFlow agent and data path controller component of the Open vSwitch software. 54, 55, 57

P4 A language to program data paths 27, 53
packet_in An OpenFlow message type. 62, 65, 66, 75
packet_out An OpenFlow message type. 62, 65
Pica8 A network management software vendor. 29, 52, 55, 56, 74
PicOS OpenFlow agent software sold by Pica8. x, 51–55, 57–65, 74, 76, 79, 80, 139, 161, 163, 165, 167, 169

resource A resource is a functional part of a hardware device that is of limited abundance, consumed through or by the execution of a program on the hardware device, and influences the output of the device. 19, 41, 42, 44, 47, 137, 138
resource controllability A resource is called controllable by the SDN controller if its operational status can be influenced either by directly controlling the resource or its workload. 48, 49, 67, 74
resource errors Resource errors are the number of error events that happened during the resource's operation [Gre13]. 49, 74
resource event An event that is caused by a resource shortage in the data plane that affects the operations of control plane applications. 83, 84, 86
resource path A resource path is the path through a resource topology a program/message/instruction moves along from the SDN controller to affect its destination resource. 42, 46, 47, 58, 67, 68, 162
resource saturation A resource is called saturated if its workload is higher than its processing capacity [Gre13]. Saturation can either be a binary state when new work is immediately dropped by the resource or a level when new work is queued for later processing. 49, 74
resource topology The graph of resources that specifies how programs/messages/instructions reach the resource they are designed to affect, starting from its source. 35, 43, 45–47, 49–51, 53, 67, 68, 81, 84, 138, 139, 162, 163, 168
resource utilization The utilization of a resource is the ratio of its used abundance to its available abundance [Gre13]. 48, 74
resource virtualization Virtualization provides controlled, shared access to a single resource from multiple consumers of the resource that have no knowledge of each other and do not necessarily behave cooperatively. 48, 50, 51

resource visibility A resource is called visible by the SDN controller if its operational status information such as its utilization, saturation, and errors are available. 48, 49, 67, 74

Trident II A switch ASIC design of Broadcom. ix, 51–53, 55, 58–60, 62, 63, 67, 76, 93, 138, 161

Trident+ A switch ASIC design of Broadcom. 76

LIST OF FIGURES

LIST OF TABLES

A APPENDIX

A.1 THE COMPLETE EDGE-CORE AS5712-54X ARCHITECTURE AND RESOURCE TOPOLOGY

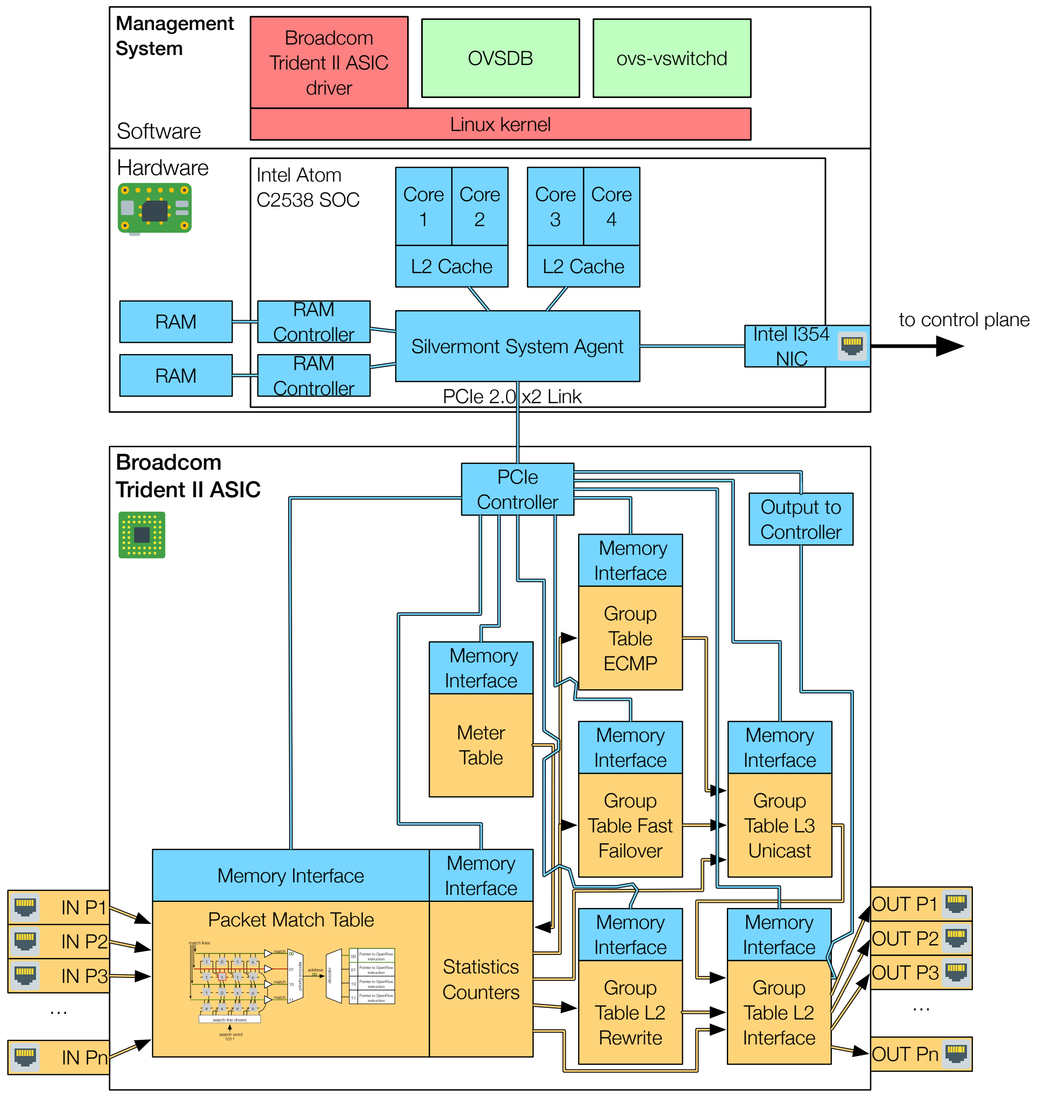


Figure A.1: The complete hard- and software architecture of the Edge-Core AS5712-54X running PicOS 2.8

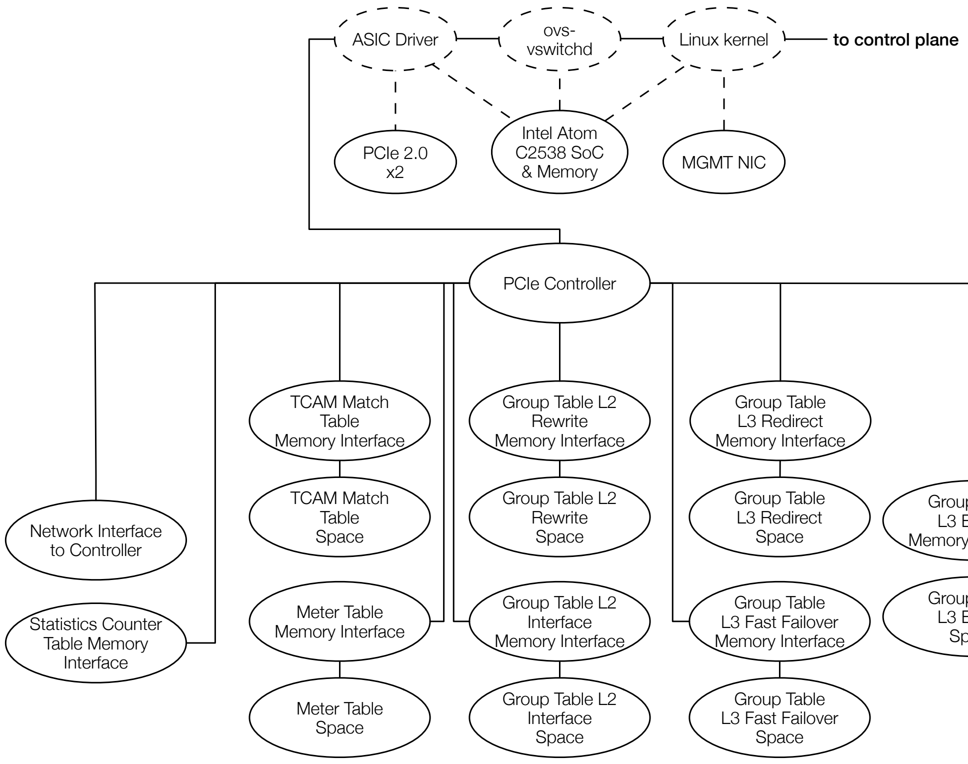

ASIC Driver
ovs-vswitchd
Linux kernel
to control plane
PCIe 2.0 x2
Intel Atom C2538 SoC & Memory
MGMT NIC
PCIe Controller
TCAM Match Table Memory Interface
TCAM Match Table Space
Group Table L2 Rewrite Memory Interface
Group Table L2 Rewrite Space
Group Table L3 Redirect Memory Interface
Group Table L3 Redirect Space
Group Table L3 ECMP Memory Interface
Group Table L3 ECMP Space
Network Interface to Controller
Statistics Counter Table Memory Interface
Meter Table Memory Interface
Meter Table Space
Group Table L2 Interface Memory Interface
Group Table L2 Interface Space
Group Table L3 Fast Failover Memory Interface
Group Table L3 Fast Failover Space

Figure A.2: The complete Edge-Core AS5712-54X resource topology as seen by the controller.

Table A.1: The complete control path resource table for the Edge-Core AS5712-54X with PicOS 2.8.

Location	**Resource**	**Metric**	**Type**	**Unit**	**Abundance type**	**Abundance**	**Allocation granularity**	**Saturation**	**Errors**
Manage-ment system	CPU cores	time	renew-able	%	static	400	no	no	no
Manage-ment system	L2 cache	space	non-renewable	MB	static	2	no	no	no
Manage-ment system	Memory	space	non-renewable	GB	static	8	no	no	no
Manage-ment system	Memory controller	time	renew-able	%	static	200	no	no	no
Manage-ment system	Silvermont System Agent	through-put	renew-able	GB/s	static	25.4	no	no	no
Manage-ment system	Management NIC	through-put	renew-able	Mp-p/s	static	~1.4	no	no	no
Manage-ment system	PCI-e link	through-put	renew-able	GB/s	static	1	no	no	no
ASIC	PCI-e controller	through-put	renew-able	GB/s	static	un-known	no	no	no
ASIC	Flow table space	space	non-renewable	en-tries	static	512	1 entry	no	yes
ASIC	Flow table memory interface	through-put	renew-able	en-tries/s	static	un-known	no	no	no
ASIC	Group table space	space	non-renewable	en-tries	static	512	1 entry	no	yes
ASIC	Group table memory interface	through-put	renew-able	en-tries/s	static	un-known	no	no	no
ASIC	Meter table space	space	non-renewable	en-tries	static	512	1 entry	no	yes
ASIC	Meter table memory interface	through-put	renew-able	en-tries/s	static	un-known	no	no	no
ASIC	Statistics counter table memory interface	through-put	renew-able	en-tries/s	static	un-known	no	no	no
ASIC	Packet output to controller	through-put	renew-able	pack-ets/s	static	>12000	meter feature	no	no

B AUTHOR'S PUBLICATIONS

MAIN PUBLICATIONS

[Ble+16a] J. Blendin, Y. Babenko, D. Kusidlo, G. Schyguda, and D. Hausheer. „Position Paper: Towards a Structured Approach to Developing Benchmarks for Virtual Network Functions." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2016.

[BH14] J. Blendin and D. Hausheer. „Towards Resource-Efficient Application-Controlled Software Defined Networks." In: *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*. Phd paper. 2014.

[Ble+16b] J. Blendin, D. Herrmann, M. Wichtlhuber, M. Gunkel, F. Wissel, and D. Hausheer. „Enabling efficient multi-layer repair in elastic optical networks by gradually superimposing SDN." In: *Proceedings of the International Conference on Network and Service Management (CNSM)*. 2016.

[Ble+15a] J. Blendin, J. Rückert, S. Bleidner, and D. Hausheer. „Taking the Sting out of Flow Update Peaks in Software-Defined Service Chaining." In: *Proceedings of the International Workshop on Management of SDN and NFV Systems (ManSDN/NFV)*. 2015.

[Ble+14] J. Blendin, J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer. „Position Paper: Software-Defined Network Service Chaining." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2014.

[Ble+15b] J. Blendin, J. Rückert, T. Volk, and D. Hausheer. „Adaptive Software Defined Multicast." In: *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*. 2015.

[Ble+18] J. Blendin, F. Bendfeldt, I. Poese, B. Koldehofe, and O. Hohlfeld. „Dissecting Apple's Meta-CDN during an iOS Update." In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. Accepted. 2018.

CO-AUTHORED PUBLICATIONS

[Ngu+17] T. A. B. Nguyen, P. Agnihotri, C. Meurisch, M. Luthra, R. Dwarakanath, J. Blendin, D. Böhnstedt, M. Zink, and R. Steinmetz. „Efficient Crowd Sensing Task Distribution Through Context-Aware NDN-Based Geocast."

In: *Proceeding of the IEEE Conference on Local Computer Networks (LCN)*. 2017.

[Nob+17] L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, and D. Hausheer. „Bare-Metal Switches and Their Customization and Usability in a Carrier-Grade Environment." In: *Proceeding of the IEEE Conference on Local Computer Networks (LCN)*. 2017.

[Rüc+15a] J. Rückert, J. Blendin, R. Hark, and D. Hausheer. „DynSDM: Dynamic and Flexible Software-Defined Multicast for ISP Environments." In: *Proceedings of the International Conference on Network and Service Management (CNSM)*. 2015.

[Rüc+16] J. Rückert, J. Blendin, R. Hark, and D. Hausheer. „Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSDM." In: *IEEE Transactions on Network and Service Management (TNSM)* 13.4 (2016), pp. 754–767.

[RBH15] J. Rückert, J. Blendin, and D. Hausheer. „Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks." In: *Springer Journal of Network and Systems Management (JNSM), Special Issue on Management of Software-Defined Networks* 23.2 (2015), pp. 280–308.

[Uni+17] N. Uniyal, D. Kutscher, J. Seedorf, J. Blendin, and D. Hausheer. „Adaptive ICN multipath forwarding for hybrid access." In: *Proceedings of the International Conference on Networked Systems (NetSys)*. 2017.

[Wic+17] M. Wichtlhuber, J. Kessler, S. Bücker, I. Poese, J. Blendin, C. Koch, and D. Hausheer. „SoDA: Enabling CDN-ISP Collaboration with Software Defined Anycast." In: *Proceedings of the IFIP Networking Conference*. 2017.

DEMO PAPERS

[Nob+18] L. Nobach, J. Blendin, H.-J. Kolbe, G. Schyguda, and D. Hausheer. „RTP packet loss healing on a bare-metal switch." In: *Proceeding of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. 2018.

[RBH13] J. Rückert, J. Blendin, and D. Hausheer. „RASP: Using OpenFlow to Push Overlay Streams into the Underlay (Demo Paper)." In: *Proceedings of the IEEE International Conference on Peer-to-Peer Computing (P2P)*. 2013.

[Rüc+14] J. Rückert, J. Blendin, N. Leymann, G. Schyguda, and D. Hausheer. „Demo: Software-Defined Network Service Chaining." In: *Proceedings of the European Workshop on Software Defined Networks (EWSDN)*. 2014.

[Wic+15] M. Wichtlhuber, F. Kaup, R. Reinecke, J. Blendin, and D. Hausheer. „Demo: A holistic energy-monitoring framework for the IT service delivery chain.“ In: *Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2015.

TECHNICAL REPORTS

[Rüc+15b] J. Rückert, J. Blendin, R. Hark, T. Wächter, and D. Hausheer. *An Extended Study of DynSDM: Software-Defined Multicast using Multi-Trees*. Tech. rep. PS-TR-2015-01. Peer-to-Peer Systems Engineering Lab (PS), Technische Universität Darmstadt, 2015.

PATENT APPLICATIONS

[BNK18] J. Blendin, L. Nobach, and H.-J. Kolbe. *Undisclosed*. Patent application. Deutsche Telekom, 2018.

C ERKLÄRUNG LAUT PROMOTIONSORDNUNG

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 1. Oktober 2018

Jeremias Georg Johannes
Lucian Blendin

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of January 21, 2019 (1.0).