Monitoring Federated Softwarized Networks

Approaches for Efficient and Collaborative Data Collection in Large-Scale Software-Defined Networks

TECHNISCHE UNIVERSITÄT DARMSTADT

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

Dissertation von Rhaban Simon Hark, M.Sc., geboren am 21.04.1989 in Köln

Referent: Prof. Dr.-Ing. Ralf Steinmetz
Korreferent: Prof. Andreas Mauthe, Ph.D.
Eingereicht am 02.07.2019
Darmstadt 2019

etit
ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Rhaban Simon Hark, M.Sc.: *Monitoring Federated Softwarized Networks,*
Approaches for Efficient and Collaborative Data Collection in Large-Scale Software-Defined Networks
Darmstadt, Technische Universität Darmstadt
Jahr der veröffentlichung der Disseration auf TUprints: 2019
URN: *urn:nbn:de:tuda-tuprints-90737*
Tag der mündlichen Prüfung: *15. August 2019*

Abstract

The term *Softwarized Networking* encapsulates technologies that allow the use of software to program a communication network. These technologies, predominantly Software-Defined Networking (SDN) and Network Functions Virtualization (NFV), have dominated the scientific interests of the networking community in the last decade. Leading companies already adopted SDN in large-scale deployments (e.g., Google's B4 Project, Microsoft Azure). According to Cisco, 76% of all data centers will apply SDN by 2021. Along with a hand full of valuable advantages, the foundation of the success of Softwarized Networking lies in its flexibility. In the case of SDN, a logically centralized controller, denoted control-plane, uses software to dynamically change how the networking devices, denoted data-plane, handle traffic. This centralization tremendously eases the management process. With respect to network state monitoring, which is a cornerstone of network management and the basis for its adaptivity, SDN provides, in addition to the advantage of the by-design centrally available knowledge, a set of new techniques to collect statistics from the networking devices.

The centralization of the control-plane quickly turned out to be only of logical nature and requires a physically distributed implementation to achieve scalability and reliability. Therefore, numerous distributed controller architectures have been proposed. Yet, the distribution of the control and, in line with this, the distribution of large-scale networks (e.g., one data center consists of a multitude of distributed sub-data centers) have not been considered in existing monitoring approaches. We believe there is a potential to increase the efficiency of monitoring when network parts collaborate.

In this thesis, we exploit this potential by developing monitoring approaches that utilize coordination and information exchange among collaborating SDN controllers. We create mechanisms to discover redundancy in the monitoring of shared resources and aggregate overlapping measurement tasks of different controllers whenever possible. Doing so, we substantially cut down the costs for monitoring, which is necessary for future networks that face a vast increase in load and dynamics. On top of this, we zoom into the statistic collection process in Softwarized Networks between controllers and the data-plane devices. Within that, we identify three not yet fully explored aspects, namely *how*, *where*, and *which* statistics to measure from the network. We propose novel methods for these aspects to collect information efficiently while limiting resource consumption. Extensive evaluations show that filtering irrelevant data can reduce the required measurement transmissions to a fraction and an intelligent measurement point placement requires only a small number of measurements compared to measuring the entire network, without affecting the accuracy.

Kurzfassung

Der Ausdruck *Softwarized Networking* fasst Technologien zusammen, welche es ermöglichen, mithilfe von Software das Verhalten eines Kommunikationsnetzes zu programmieren. In den letzten Jahren haben ebensolche Technologien, insbesondere Software-Defined Networking (SDN) und Network Functions Virtualization (NFV), das wissenschaftliche Interesse dominiert. Führende Unternehmen nutzen seitdem zunehmend SDN für große Netze (Google B4, Microsoft Azure). Cisco beziffert den Anteil der Datenzentren, die im Jahr 2021 voraussichtlich SDN nutzen werden, auf 76%. Neben vielen weiteren Vorteilen ist die gewonnene Flexibilität von Softwarized Networks Grundlage des Erfolgs. Im Fall von SDN definiert ein logisch zentraler Controller (Control-Plane) dynamisch, mithilfe von Software, wie das Datennetz (Data-Plane) Pakete verarbeitet. Diese Zentralisierung erleichtert viele Aspekte des Netzwerkmanagementprozesses. In Hinblick auf die Statusüberwachung bietet SDN einige praktische Techniken, um Statistiken aus Netzwerkelementen abzufragen. Die effiziente Überwachung des Netzes stellt eine Kernkomponente des Netzwerkmanagements und Grundlage für dessen Adaption dar.

Die Zentralisierung der Control-Plane ist nur von logischer Natur und muss physisch verteilt implementiert werden um benötigte Skalierbarkeit und Zuverlässigkeit zu gewährleisten. Hierfür wurde bereits eine Vielzahl von Architekturen vorgeschlagen. Im Gegensatz dazu wird die Verteilung der Control-Plane sowie die Verteilung von großen Netzen (z.B. verteilte Datenzentren) bei existierenden Monitoringansätzen nicht berücksichtigt. Entsprechend ist der erste Beitrag dieser Arbeit, Ansätze vorzustellen, um mithilfe der Koordination kollaborierender Netze dieses Potential zu nutzen und somit die Monitoringeffizienz zu erhöhen. Die entwickelten Ansätze erkennen redundante Messungen von gemeinsam genutzten Netzressourcen und aggregieren überlappende Monitoringaufgaben verschiedener Controller. Mithilfe dieser Ansätze werden die Monitoringkosten stark reduziert und können dadurch die erwartete ansteigende Last und Dynamik zukünftiger Netze bewältigen.

Weiterhin fokussiert sich diese Arbeit in einem zweiten Beitrag auf den Prozess der Statistiksammlung in der Beziehung zwischen Controllern und Elementen der Data-Plane. Dabei werden drei Aspekte identifiziert, die vom aktuellen Stand der Forschung nicht vollständig abgedeckt sind: *Wie* und *wo* müssen *welche* Statistiken aus dem Netz abgefragt werden. Hierzu werden neue Methoden vorgestellt, welche diese Aspekte behandeln, indem möglichst effizient Informationen bei geringem Ressourcenverbrauch gesammelt werden. Eine umfangreiche Evaluation zeigt, dass z.B. das Herausfiltern von irrelevanten Messwerten eine Reduktion von zu übermittelnden Messwerten auf einen Bruchteil erlaubt. Weiterhin wird gezeigt, dass eine geringere Anzahl an Messungen die gleiche Genauigkeit liefern kann, wenn eine intelligente Messpunktplatzierung, wie in dieser Arbeit vorgeschlagen, genutzt wird.

Danksagung

Für die erfolgreiche Anfertigung einer Dissertation braucht es mehr als nur den Verfasser. Viele Menschen haben Anteil daran, dass der Weg dorthin möglich und das Ziel erreicht werden konnte. Ich hatte das große Glück, von vielen Seiten gutmütig unterstützt worden zu sein und hilfsbereite Menschen an meiner Seite zu haben. Ebendiesen Menschen möchte ich dafür von Herzen danken.

Zuerst gebührt denjenigen Dank, die fachlich und organisatorisch unmittelbar für den Inhalt und die Umsetzung der Dissertation essentiell waren: Ralf (Steinmetz), dem ich die Möglichkeit zur Promotion neben der Anstellung als wissenschaftlicher Mitarbeiter in den letzten vier Jahren mitsamt des hervorragenden Arbeitsklimas verdanke. Darüber hinaus hat insbesondere mein Betreuer der letzten dreieinhalb Jahre, Amr, einen großen Anteil an der erfolgreichen Anfertigung der Disseration. Neben der direkten fachlichen Unterstützung hinsichtlich meiner Forschung habe ich unglaublich von seinem Wissen und seiner Erfahrung profitiert – danke! Einen großen Beitrag hat auch Dominik geleistet, der am Anfang meiner Zeit als wissenschaftlicher Mitarbeiter der wichtigste Anlaufpunkt war und mit dem die Grundzüge meiner Forschung entstanden sind. In diesem Zug möchte ich auch Julius danken, der bei mir die Lust für das Thema geweckt hat, bei dem ich im Masterstudium diesbezüglich viel gelernt habe und der durch die intensive Betreuung während der Masterarbeit die Türen für die Promotion bei KOM geöffnet hat. In direkter Verbindung mit der Dissertation möchte ich auch den Vielen danken, die sich Zeit genommen haben um Teile der Ausschrift zu lesen und Feedback zu geben: Amr, Anam, Vera, Claudi, Mohamed, Khalil, Patrick, Ralf, Nils und der Serious Games Task-Force.

Neben der inhaltlichen Unterstützung ist natürlich die Arbeitsumgebung mitsamt ihrer Atmosphäre ein integraler Bestandteil der Promotionsphase. Ich bin sehr glücklich, dass ich an unserem Fachgebiet viele nette Menschen an meiner Seite habe, die diese Zeit begleitet und gestaltet haben. Hierbei gilt zunächst Boris besonderer Dank, der als Gruppenleiter immer unterstützt, wo es geht. Die Liste der Kollegen, die den Weg unterstützt haben ist zu lang, um alle zu benennen. Trotzdem möchte ich einige hervorheben. Dazu zählen zunächst die Kollegen aus der AOC Gruppe, mit denen ich tagtäglich interagiere: Ralf (Kundel) – immer gut für witzige und ironische Smalltalks, aber auch fachlich sehr hilfreich; Manisha – symphatisch und immer hilfsbereit; Sounak – ein sehr angenehmer, ruhiger Büropartner; die Ehemaligen wie Jeremias, Rahul, Alex Frömmgen, Denny Stohr – alles angenehme Menschen mit gutem Humor. Besonders wichtig ist auch Sonja, mit der ich viel gute, lustige Zeit in einem Büro verbracht habe und von deren Erfahrung ich vielfach profitiert habe. Dann natürlich Nils, mit dem ich meine politisch nicht immer korrekte Ader ausleben durfte und der ein guter Freund geworden ist (obwohl er Abends nie Zeit hat weil er immer Klettern ist). Neben den Kollegen in der AOC Gruppe möchte ich allen KOM'lern, etwa Tobi, Thomas, Binh, . . . , für die gute Zeit und Unterstützung danken. Speziell auch Björn, dessen Feedback immer mehr als wertvoll war. Mein besonderer Dank gilt auch Britta, weil wir neben der Arbeit gute Freunde geworden sind und Du den Arbeitsalltag belebt hast. Danke für die unzähligen Mittagsessen, Tees, Kaffees, Autofahrten, . . . – man findet nicht häufig Menschen, mit denen man in jeglicher Hinsicht auf einer Wellenlänge ist. Gleiches gilt für Patrick: New York, Chicago, Köln bleiben unvergessen – hätte nicht besser sein können – mehr wird folgen. Dank auch an nicht-KOM'ler Kollegen, z.B. aus MAKI, mit denen ich gute Erfahrungen gemacht habe. Hierbei in letzter Zeit insbesondere Anam, die sich Zeit genommen hat, meine Ausschrift zu lesen und eine lustige Kaffeepartnerin ist. Dann möchte ich mich bei meinen Studis bedanken: Mohamed, Khalil und Taushif, drei intelligente Menschen, inzwischen Freunde, deren Beiträge weitreichend in die Dissertation eingeflossen sind. Ich danke Michael Zink für die Einladung nach Massachusetts und

die Unterstützung, insbesondere aber nicht ausschließlich in dieser Phase. Dank an Michael's Gruppe, vor allem Divya und Raj, die mich sehr freundlich aufgenommen haben und mir in Amherst Vieles gezeigt haben.

Für den guten Arbeitsalltag sind auch die ATMs nicht wegzudenken, denen wir als wissenschaftliche Mitarbeiter viel verdanken: Sabine, Karola, Frau Ehlhardt, Stephan (Tittel), Frau Scholz-Schmidt – die alles weiß und auch eine angenehme Mittagsessenbegleitung ist – sowie Frank und Moni, mit denen die Adminarbeit effektiv und mit guter Laune abläuft. Dazu zähle ich auch Zeynep, die sich sehr für mich eingesetzt hat und der ich viel verdanke. Zuletzt auch Julia und Michaela, dank derer ich in Zukunft mit einem heiteren, wohlwollenden MAKI-Office interagieren werde. Weiterhin möchte ich meinem Mentor Johannes Konert danken, der meine erster Berührungspunkt mit KOM war, bei dem ich als HiWi mehrere Jahre viel lernen konnte.

Es ist gut, auch abseits der Arbeit Menschen zu haben, die einen unterstützen und für den nötigen Ausgleich sorgen, so dass die Dissertation positiv verlaufen konnte. Auch hier gibt es so viele, dass ich nur ein paar benennen kann: Paul, mit dem ich weit mehr teile als das gemeinsam verbrachte Studium. Die *Taka-Tuka Ultras*: Paul, Sebastian und Lukas – danke für die feuchtfröhliche Zeit in *der WG*. Auch mit Lea durfte ich in *der WG* wohnen, die uns zu einer anhaltend engen Freundschaft verholfen hat. Heinrich, Leif, Nadja, Marius, Rebecca Kim und Raphael, mit denen ich schon so lange so gut befreundet bin und hoffentlich noch ewig sein werde. Flo mit Dori und den wunderbaren Fiete und Simon, mit dem ich eine einzigartige Freundschaft teile, trotz der vielen Kilometer und der wenigen Zeit. In die Kategorie "Menschen, die mir wichtig sind" zählen auch Nauka und Ruben – danke für die vielen Jahre, in denen Ihr mir gute Freunde wart und sein werdet.

Dann bleiben die Menschen, die mir am nächsten sind. Ich danke dir, Claudi, dafür, dass Du so vieles auf Dich nimmst. Mich drei Monate nach Amerika lässt und mich dabei unterstützt auch wenn es Dir schwer fällt. Danke, dass Du mich motiviert hast zu schreiben, wenn wir stattdessen auch schönere Dinge hätten machen können und danke, dass Du meine häufige Abwesenheit tolerierst und trotzdem immer für mich da bist. Zuletzt danke ich meiner großartigen Familie, Manuel mit Anne, Fiona und Finn sowie Vera mit Arnt – Geschwister und Verwandte, zu denen ich aufblicken kann, auf die man Stolz sein kann. Und an erster Stelle meine Eltern. Es ist unglaubliches Glück, ohne Wenn und Aber bei allen Entscheidungen, auch in Hinblick auf Studium und Promotion, unterstützt zu werden, ohne Druck und Erwartungen. Noch dankbarer bin ich darüber, dass sich bei uns niemand Sorgen machen muss, irgendwann alleine da stehen zu müssen. Ich danke Euch dafür, dass Ihr mich gelehrt habt, mir meine eigenen Gedanken zu machen und stets gelassen zu sein. Ich danke Euch für alles, was Ihr mir beigebracht habt. Darauf baut alles auf was ich bin. Ich danke Euch für die Gutmütigkeit und Liebe.

Ich danke auch denen, die mich in den letzten Jahren auf irgendeine Art und Weise unterstützt haben, die ich nicht erwähnt habe – es werden einige sein, seht es mir nach. Zum Abschluss habe ich keinen schlauen Spruch darüber wie wichtig das moderne Kommunikationsmöglichkeiten für uns alle ist (obwohl sie das geworden sind) - möchte stattdessen ganz im Gegenteil darauf hinweisen, dass es wichtigere Dinge gibt als den Fortschritt und Erfolg. Und nehmt Euch nicht immer ernst ☺.

"More than machinery, we need humanity.
More than cleverness, we need kindness and gentleness."
– Charlie Chaplin (The Great Dictator, 1940)

Content

1

Introduction

The scientific interest in Software-Defined Networking (SDN) strongly increased since its concept was enlivened at Stanford University [McK+08] in 2008. Besides great attention in the scientific community, SDN also became an essential keyword for networking companies, which adopted the principle in their productive networks (e.g., Microsoft's Azure data centers [Gre15] and Google's B4 Project [Jai+13]). As a consequence, key players in the communications market formed important consortia, such as the Open Networking Foundation[1], fostering SDNs. Cisco predicts that in 2021, around 67% of data centers will partially or fully deploy SDN while assessing the portion in 2019 already to 43%[2]. In total 7,4 Zettabytes will be delivered in 2021 using softwarized technologies, which is a traffic share of 50% *within* data centers [Cis18]. The need for a new network architecture originated from the inability of today's rigid network infrastructures and dependency on established protocols to handle the tremendously increasing amount of traffic and its service requirements other than with excessive and cost exhaustive over-provisioning. On top of this, the dependence on vendors, which built specialized machines for various purposes[3], heavily limited the flexibility of network providers. Hence, both the dependency on vendors and long-established protocols halted the potential for innovation. To overcome this, the SDN paradigm decouples the control (control-plane) from the forwarding (data-plane) in communication networks. Software applications running on a logically centralized controller dynamically control how switches in the network behave [LLC15] by using open protocols, such as *OpenFlow* [Sta13; Pfa+12]. This *network programmability* brings enormous advantages, such as the support to easily deploy new protocols. While enabling fast innovation, SDN still provides stability for productive traffic, which uses the same physical resources [Ope12; McK+08]. Moreover, SDN provides the potential to flexibly and quickly adapt the network using software, which can then fulfill changing service demands. SDN controllers benefit from a global view on

1 The Open Networking Foundation Board members include Deutsche Telekom, AT&T, China Unicom, Comcast, and Google; `http://opennetworking.org/board`, accessed 30 May 2019.

2 Statistics were last updated on 19 November 2018.

3 e.g., Switching Hardware: *Broadcom's Fibre Channel Networking Switches*; `http://broadcom.com/products/fibre-channel-networking/switches`, accessed 11 June 2019, Firewalls: *Cisco's FirePOWER series*; `http://cisco.com/c/de_de/products/security/firewalls`, accessed 10 June 2019, and Intrusion Detection Hardware: *Juniper Advanced Threat Prevention Appliance*; `http://juniper.net/de/de/products-services/security/advanced-threat-prevention-appliance`, accessed 10 June 2019.

the network and simple, yet powerful techniques to observe its state in real time. In addition to SDN, the pursue of programmability in networks brought up the key technologies *Network Functions Virtualization (NFV)* [NFV12] and, lately, *programmable packet processors*, namely *P4* switches [Bos+14]. All of these innovations set up a new era of networks described with the collective term *Softwarized Networks*. Jointly, they allow network operators to define and configure, i.e., to *program*, the network according to individual needs and to flexibly alter its behavior, if necessary in a minimum of time.

Moreover, network state monitoring constitutes a cornerstone of the network management, particularly concerning dynamic demands future networks will face. To exploit the possibility of SDN to change a network's behavior quickly, operators steadily require up-to-date state knowledge in combination with current information on pending demands. Ultimately, reliable network monitoring is crucial ever since communication networks exist to ensure stability and fail-safety by reacting to changing circumstances.

1.1 Motivation and Research Gap

Due to the new SDN architecture and, therewith, the emergence of new measurement techniques, the field of monitoring in SDN faces a broad potential for improvements. The use of existing techniques from legacy networks, based on NetFlow [Cla04], sFlow [PPM01], SNMP [WHP99], and the alike remain applicable. However, SDN mechanisms such as OpenFlow's packet/byte counter [Pfa+12] can replace many aspects in an easy and resource-friendly manner. In this line, numerous studies propose intelligent approaches, designed for SDN, to effectively obtain a network state representation in the control-plane. The related works include spatial [JYR11; Zha13] and temporal [vDK14; Cho+14c] statistic granularity adaptation, memory-efficient statistic records on the data-plane [YQL14; YJM13; YJM13], SDN-tailored state estimation applications [SBB13; TGG10; Gio+14], etc.

Nonetheless, quickly after the rise of SDN, it became clear that the idea of a centralized controller is not feasible with physical centralization. Since a central entity introduces a single point of failure; thus, it is vulnerable to controller faults as well as being not scalable, distributed control-planes became state-of-the-art. The logically centralized controller itself is a network of multiple interconnected physical controllers. These controllers share responsibilities with respect to their location and workload. Again, numerous studies propose architectures and paradigms on the organization of distributed control-planes. The wide spectrum of approaches ranges from hierarchical architectures [HG12] to works that focus on the exchanged information between controllers [PBL14] to approaches that dynamically spawn and relocate controllers [Dix+14; SSC15]. Besides distributed control-planes, most large-scale networks nowadays are organized as a federation of multiple sub-networks. For instance, enterprise and campus networks unite and inter-

connect smaller networks in different locations. Moreover, data centers are distributed in order to push information closer to users. Still, they can be understood as one large data center.

In view of today's and future networks, which consist of multiple subnetworks controlled by a network of controllers, we identify a lack of monitoring approaches that consider the actual SDN architecture. Most monitoring approaches abstract the controller as a single entity, while its actual implementation is left unconsidered in the approaches. Other approaches ignore the distributed nature of SDNs and bundle all load on one controller. We state that collaboration among cooperative network parts of a federated network has a huge potential to increase the efficiency of network state monitoring, which is necessary with regard to the steadily increasing demands and load on the network. We use the abstract term *efficiency* to describe the relationship between performance, e.g., measurement accuracy, and costs, e.g., message overhead. Providing controllers the possibility to exchange monitoring related information, particularly for networks with shared common resources, such as connections and traffic, offer the opportunity to enrich each other's information and avoid unnecessary redundancy. This raises the first research question for the work in hand:

Research Question 1: *How can the efficiency of monitoring be increased in view of the distributed control and architecture of collaborating federated Software-Defined Networks?*

Moreover, in this thesis, we take a closer look into the statistic information collection process in the context of Softwarized Networks. Here, we find a plethora of useful approaches as sketched earlier. Within that, we identified three core aspects that are not yet fully explored and seemingly have the potential for efficiency improvement, which we tackle in this thesis: *(i)* First, SDNs provide a large set of techniques to gather information from the data-plane. In combination with classic collection techniques, the set becomes even larger. Following the notion that techniques are superior and inferior depending on the network conditions and monitoring requirements, it is clear that there is no single technique that can be favored in every case. Consequently, an open question is *how* to reconcile this variety of possibilities. *(ii)* Second, the question of *where* to gather information is challenging. Existing works focus on measurement point placements using constraint-based optimization techniques or heuristics approximating them. As such approaches require high computational resources and time for revaluation, they are not suitable for future networks with highly dynamic load distribution. Consequently, monitoring lacks strategies to quickly place measurement points by leveraging the centralized knowledge available in SDNs. *(iii)* Last, within the collection process, studies mainly focus on how to transmit information from the data-plane to the control-plane. A sensible

selection of information concerning a measurement's information remains insufficiently investigated in the context of SDN. We regard this aspect as vital to be tackled in order to cautiously invest costs for network state monitoring and its analysis. These three gaps build the foundation of the second research question:

Research Question 2: *How and where must information be collected and selected to efficiently monitor the state of dynamic Softwarized Networks?*

1.2 Research Goals

The essential goal of this thesis is to explore methods, which increase the efficiency when monitoring the state of federated Software-Defined Networks controlled by multiple cooperative controllers. This superior goal summarizes two research goals, formulated in the following, that intend to answer the research questions. The second research goal pools three subordinated subgoals. Figure 1 illustrates them in an abstract high-level view on federated SDNs. In the upper layer, the figure shows three exemplary controllers that are interconnected. Each controller manages one part of the network, depicted as dashed connections between the controller and the corresponding switches on the infrastructure layer (yellow).

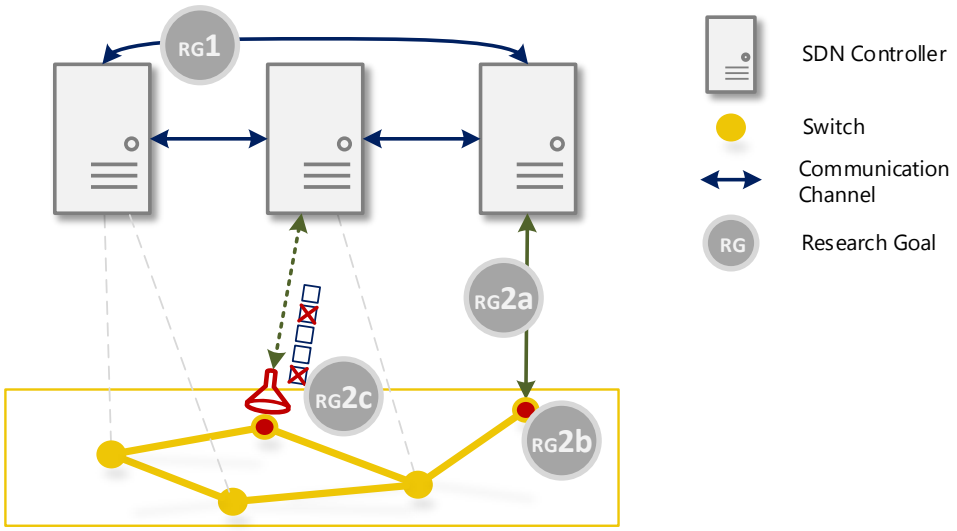


Figure 1: Illustration of the research goals within a federated SDN architecture with distributed control-plane. Whereas the first research question mainly concentrates on the control-plane interconnection, the second research question, displayed with three subgoals, focuses on the data collection process between the data-plane and control-plane.

Research Goal 1: *Design and evaluate mechanisms for collaborative monitoring to take advantage of the distributed control and architecture of collaborating federated SDNs.*

This goal mainly refers to the collaboration on the distributed control-plane, as marked with **RG1** in the figure, and is closely related to the first research question. We intend to develop mechanisms for distributed control-planes to share information and coordinate monitoring applications to save costs. By enabling collaboration, we seek to reduce overhead for measurements that concern shared resources of multiple controllers. To give an example, two controllers must not redundantly observe a connection between two adjacent network parts. Finally, we want to prove our proposed concept, as described in the following chapters, and show the efficiency gain of the methods in an exhaustive evaluation.

Research Goal 2: *Design and evaluate methods, which increase the efficiency of the monitoring data collection process in dynamic Softwarized Networks.*

The second goal tackles the second research question and is threefold. According to the second research question, the three subgoals each tackle an aspect of the collection process: *(a)* The first subgoal aims to develop techniques, which capture information from the data-plane elements. Thus, *how* to capture statistical information. Sketched with **RG2a**, numerous techniques to gather statistics exist. Therefore, we seek to find a method, which achieves good accuracy while maintaining a low-cost profile in all situations. *(b)* The second subgoal, depicted with **RG2b** in Figure 1, tackles the measurement point selection, hence, *where* to capture statistical information. We want to obtain as much information as possible by intelligently placing measurement points. This goal mainly concerns the trade-off between a measurement's informativeness and its costs. *(c)* The last subgoal within the data collection process is, generally speaking, again to improve the efficiency while considering the informativeness of each measurement and its costs. This subgoal focuses on the question *which* statistical information to capture as not all requested statistics prove valuable information. Sketched with **RG2c**, we intend to dismiss statistics that introduce a disproportion between the quality of information and the overhead. Although all three aspects strongly interweave within the collection process, we aim to design the mechanisms such that they can be applied independently. Consequently, we will evaluate the principles for each subgoal individually.

1.3 Structure of the Thesis

The thesis is structured in four further chapters plus appendices. First, in Chapter 2, we clarify backgrounds necessary for the understanding of the

contributions. We presume basic knowledge on communication networks, including the Internet layers and distributed systems. On top of this, we provide brief overview on *Softwarized Networks*, which include the new generation networking technologies Software-Defined Networking, Network Functions Virtualization, as well as programmable data-planes. The background also introduces federated networks to set up the thesis' main scenario and provide use cases. In addition, Chapter 2 comprises the state-of-the-art, which further elaborates the identified research gaps and gives a broader overview of related approaches. The subsequent Chapter 3 contains the contribution, which intends to achieve the first research goal. In this chapter, we propose mechanisms for collaborative monitoring in federated SDNs. It shows the design of three progressive mechanism designs. Moreover, it includes the evaluation of their efficiency and performance in, first, a synthetic scenario for a proof of concept and, second, a distributed data center scenario with respect to applicability. Chapter 4 comprises the contributions, which aim to achieve the second research goal. It starts with a mechanism to increase the monitoring efficiency by carefully leveraging measurement techniques. Afterwards, we look into the placement of measurement points and, finally, show our contribution towards the selection of measurements to increase the ratio between informativeness and costs. Each of the three contribution sections separately contains its evaluation results. We summarize the thesis and recap the contributions given the defined goals in Chapter 5. This chapter also compiles the outlook on opportunities to continue this work.

2

Background and State-of-the-Art

This chapter is comprised of a brief background on technologies such as Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) pooled as *Softwarized Networking*. In addition, the chapter shows recent advances in those technologies with respect to the contributions of this thesis. It thereby gives a brief overview of related work and shows more detail of work directly connected to our contributions. The goal of this chapter, apart from general background and state-of-the-art overview, is to highlight the lack of solutions for the identified problems.

2.1 Softwarized Networking

We use Softwarized Networking as a collective term for emerging technologies nowadays renowned as the future of networking. The term describes that networks are based on software - thus, they are programmable. Two central technologies shape this trend predominantly: *Software-Defined Networking* and *Network Functions Virtualization*. The following subsections briefly motivate and introduce both technologies. Furthermore, a subsequent subsection describes a recently emerging trend of programmable forwarding elements.

2.1.1 *Software-Defined Networking*

This section introduces the Software-Defined Networking (SDN) paradigm with a historical background, its architecture, the control layer architecture, and the most popular SDN-implementing protocol, denoted *OpenFlow*.

The Open Networking Foundation denoted SDN as the new norm for networks in the title of their white paper in 2012 [Ope12]. Although the claim was reasonable at that time [Sta13], the main aspects of its general concept were proposed much earlier. In 1982, Horing et al. [Hor+82] introduced "program controlled networks" and so-called active networks, which were coined around 1990, proposed the idea of programmable networks [Ten+97]. In the 2000s approaches such as SANE [Cas+06] and Ethane [Cas+07] followed this idea and Greenberg et al. proposed a "Clean Slate 4D" approach [Gre+05] dividing the network into multiple layers with different responsibilities. Enlivening these ideas, the Software-Defined Networking paradigm finally gained massive popularity after the OpenFlow [McK+08] standard, for the communication between the control- and data-plane was published by the Stanford

University in 2008. Subsequently, leading network vendors fostered SDNs by adopting the paradigm [Jai+13; Gre15] and advertising with "OpenFlow-enabled" devices around 2013[4,5]. As a consequence, a plethora of works focused on SDN in the last decade, which propose various advancements towards network programmability.

The primary motivation for a new norm of network architecture lies in the significant drawbacks of today's legacy networks. The fully distributed architecture became rigid as every device abides by the well-defined but outdated protocols partially developed decades ago. This static environment, combined with strongly vendor-dependent devices and a vast increase in network load led to challenging scalability constraints. Being able to centrally program the network consequently eases *(i)* the development of innovative protocols, *(ii)* the implementation of network functionality, and *(iii)* the unprecedented flexible network management. Programming new protocols into the network is an effective way to avoid rigid behavior due to missing functionality. Moreover, SDN allows network slicing so that a network can evaluate the functionality of new protocols without disturbing productive traffic. SDNs can easily realize simple network functions into the network without the need for dedicated, expensive hardware. Simple firewalls are a paramount example of such functions. Lastly, the network management also becomes flexible. Centralized controllers can change the network's behavior in a fraction of the time required so far, without the need for slow converging, distributed algorithms.

Software-Defined Networking Architecture

The central paradigm introduced by SDN is its architecture. Instead of a planar network of devices organizing themselves using distributed algorithms, SDN organizes the network in two planes upon each other: the control- and the data-plane. Figure 2 sketches the logic layers in these planes. The control-plane can be divided into an application and a control layer. Within that, the control layer consists mainly of logically centralized controllers, which comprise the intelligence. Furthermore, the application layer holds an arbitrarily large number of applications, each with a specific function, such as routing, access control, security enforcement, etc. Both layers are connected through the controller's northbound interfaces. The applications use the underlying controllers as a bridge to compile their functionality to the network infrastructure. Therefore, often there is only a vague distinction between both layers in practice. Overall, the control-plane's sole purpose is the management of the data-plane. In contrast, the data-plane consists of infrastructure devices,

4 Steven Levy, *WIRED*, "Going with the flow: Google's secret switch to the next wave of networking," 2012 [Online] https://www.wired.com/2012/04/going-with-the-flow-google/, accessed 12 February 2019

5 Jim Duffy, *Network World*, "NEC rolls out OpenFlow for Microsoft Hyper-V," 2013 [Online] https://web.archive.org/web/20130403025856/http://www.networkworld.com/news/2013/012213-nec-openflow-266024.html, accessed 12 February 2019

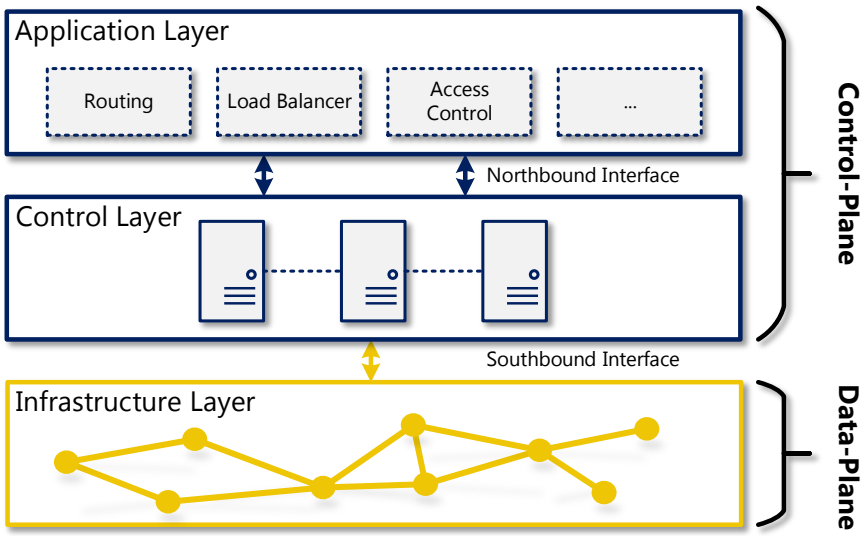


Figure 2: Software-Defined Networks split into the control- and data-plane. The data-plane contains all elements that interact with productive traffic. The control-plane controls the data-plane, thus includes all management units. It can be further divided into control layer and management application layer[6].

majorly switches, and all other elements encountering and processing production traffic. Dividing the network into separate layers and, particularly, moving the intelligence to a control-plane entails significant advantages compared to legacy networks. The ONF SDN whitepaper [Ope12] provides a profound collection, including those already noted in this chapter.

Distributed SDN Control-Planes

SDN intends to give all responsibilities to centralized controllers. The controller has, therefore, a bird's-eye view on the whole network and gathers all potentially required information in a single entity. Using a centralized entity strongly eases the network management process, distributed routing algorithms, for instance, become simple shortest path algorithms. Nevertheless, using centralized entities involves several well-known drawbacks, particularly in terms of scalability and fail-safety [Tun+15; Lev+12]. Coherently, there is a broad consent that the control-plane is logically centralized but not necessarily implemented as a single physical entity. To date, there is no standardized solution for the architectural structure or the communication among controllers of distributed control-planes. Yet, the networking research community proposes many approaches to organize distributed control-planes. Sec-

6 Note that there are different definitions of layers in the SDN architecture in literature.

tion 2.2 surveys these existing approaches as this field is of special interest for the key scenarios considered in this thesis.

OpenFlow Protocol

OpenFlow [McK+08] deserves major credit for the recurring popularity of centrally controlled networks. This open protocol specifying the communication between controllers and switches in an SDN as well as the behavior of switches themselves (cf. Figure 3) became the widely accepted de-facto standard[7] for software-defined Ethernet network management [Lim12].

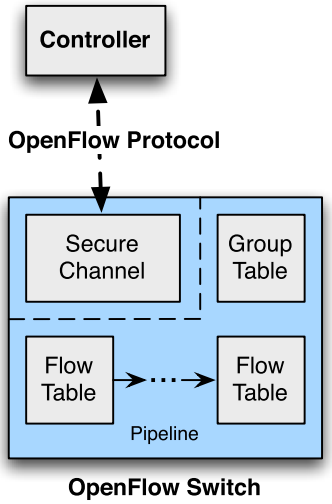


Figure 3: OpenFlow defines the protocol between the controller and OpenFlow switches. Furthermore, it specifies the switch behavior and table structure; however, not its implementation [Pfa+12].

The core of the protocol relies on the *match-action paradigm*. OpenFlow switches match packets to entries in a so-called flow table populated by the controller based on header fields. If it finds matching entries, it selects the entry with the highest priority and executes the actions associated with the entry. The size of flow tables is usually limited as vendors implement them using expensive and energy consuming ternary content-addressable memory (TCAM). Using TCAM allows rapid entry lookups in constant time [NKS00]. Nevertheless, it is common practice to leverage wildcard-based matching to, e.g., use longest prefix matching, which efficiently saves memory. If the switch cannot match a packet to a table entry, it typically sends it to the controller, which decides how to proceed. A table entry contains a priority, actions, and counters for the number of bytes, packets, and lifetime of each entry as depicted in Table 1. Controllers commonly use the counter fields

7 Its acceptance decreased with the advent of P4 in 2014 due to the inflexible and limited header specification possibilities in OpenFlow [Bos+14].

to monitor traffic on flow-level. Such counters are also available per port so that controllers can additionally monitor traffic on link-level. The latest published version [Pfa+15a] supports pushing flow counter proactively to the controller when reaching configurable thresholds. Lastly, a flow table entry holds an idle- and hard-timeout to discard outdated flow rules and a cookie as entry identifier.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table 1: OpenFlow flow table structure [Pfa+12].

Controllers can define and parameterize actions out of a predefined set of actions. These include, for instance, dispatching a packet through a port, dropping packets, duplicating packets, modifying header information, sending packets to the controller, and so on. The entire set of available actions in OpenFlow 1.3 is listed in Appendix A.1. Actions can either be applied directly or stored for later application.

OpenFlow also defines the communication protocol between controllers and switches through a secured channel. The OpenFlow specification [Pfa+12] provides a detailed overview of the protocol and switch specification. However, OpenFlow's specification does not define the behavior of controllers. In addition, it does not specify how the application layer (cf. Figure 2, p. 9) interacts with controllers. Accordingly, a variety of implementations exists. A number of implementations, such as Floodlight[8] or RYU[9] allow direct integration of applications in the controller as modules. Apart from sophisticated, mostly open source software-only controllers (e.g., Floodlight[8], RYU[9], OpenDaylight[10], NOX OS[11]) that run on legacy hardware, professional, closed source hardware controllers (e.g., NEC UNIVERGE® PF6800[12]) are also available. OpenFlow has built-in support for multi-controller environments. In contrast to the previously described distributive organized control-planes, OpenFlow considers one-to-many connections for switches. A switch can connect to multiple controllers while controllers take roles such as *Master*, *Slave*, or *Equal*. *Master* controllers exist only once within a network and forces other controllers to be *Slaves*. As *Slave* controller no write operations are possible; nevertheless, reading a switch's state is possible. Controllers in the *Equal* state have the same access rights as a *Master* controller; still, there can be multiple controllers in an *Equal* role.

In this thesis, we predominantly rely on OpenFlow as an exemplary control protocol. We assume that it's functionality is representative of any other and

8 Floodlight Project `http://www.projectfloodlight.org/floodlight`, accessed 13 February 2019
9 RYU SDN Framework `https://osrg.github.io/ryu`, accessed 13 February 2019
10 OpenDaylight Platform `https://www.opendaylight.org`, accessed 13 February 2019
11 NOX Network Control Platform `https://github.com/noxrepo/nox`, accessed 13 February 2019
12 NEC PF6800 `https://www.necam.com/sdn/Software/PF6800UnifiedNetworkCoordinator`, accessed 13 February 2019

future protocols with the same purpose. If not stated differently, we design all approaches in a way that any other protocol could be used instead and do not require OpenFlow specific properties.

2.1.2 *Network Functions Virtualization*

Network Functions Virtualization (NFV) is the second fundamental pillar of Softwarized Networking. An ETSI consortium [NFV12] coined and defined the basic concept in 2012 in Darmstadt, Germany. The main idea behind the concept is replacing vendor-dependent fixed hardware appliances that are tailored towards only a single network function with software-based virtual appliances running on commodity hardware. Due to an increase in the size and complexity of networks, the need for specialized functions increases likewise. However, fixed hardware with high costs and maintenance overhead as well as vendor-dependency hinders the development and, ultimately, the development of new services. As a solution, NFV targets to virtualize network functions and flexibly deploy them as software pieces onto standard commodity servers. In combination with SDN, NFV enables network operators to design, develop, and evaluate innovative network services [Han+15]. Moreover, NFV supports quick and flexible scaling of services to address changing demands. Virtually every function can be implemented and deployed as a Virtual Network Function (VNF): Firewalls, switches, intrusion detection, load-balancing, etc.

2.1.3 *Programmable Data-Planes*

In 2014, Bosshart et al. [Bos+14], pointed out that there is a lack of flexibility with respect to non-established protocol headers of current prevalent technologies. As OpenFlow supports a large, yet finite set of matchable header fields, limiting the openness for new protocols and services, and there is no opportunity to alter behavior on any of the network layers, they propose a general solution to make switches programmable. The introduced *P4* (Programming protocol-independent packet processors [Bos+14]) language rapidly gained acceptance in the networking community. As a result of its broad acceptance, P4-programmable Ethernet switching ASICs, such as the Barefoot Tofino family[13], supporting early versions of P4, became available.

The main principle of P4 is the programmability of the behavior of forwarding switches that OpenFlow defined explicitly beforehand. Programming the whole processing pipeline allows network operators to specify custom header fields, match them in line-rate and program custom actions to be performed. Basic examples include scalable load-balancing schemes [Kat+16], network assisted video streaming [Bha+17], and heavy hitter detection entirely on

13 Barefoot TOFINO2 https://www.barefootnetworks.com/products/brief-tofino-2, accessed 14 February 2019

switches [Siv+17]. P4's syntax partly looks similar to C syntax. According to Figure 4 and P4's Language Tutorial[14], it contains code for *(i)* all header definitions, *(ii)* the parser behavior with bit-extractions, *(iii)* multiple pipeline steps of matching and corresponding actions, and *(iv)* a deparser.

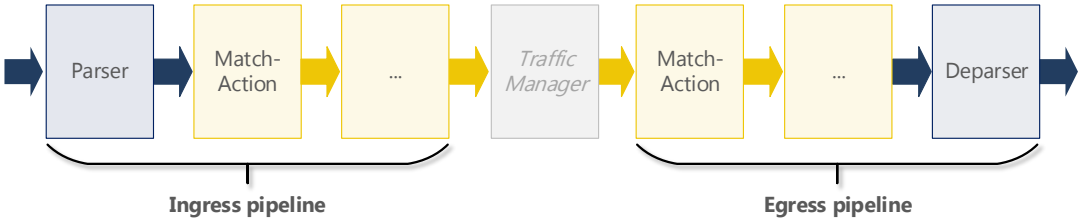


Figure 4: P4's simplified programmable switch processing pipeline model: The first pipeline element is a programmable packet parser. Subsequently, multiple match-action controls and a traffic manager for packet replication and egress queue management. Afterwards, further match-action controls and a deparser before the packet reaches the egress ports.

A vendor-supplied compiler reads the P4 program and compiles it for an architecture model target. The target-specific binaries can then be uploaded to the corresponding P4 device. Note that different targets might support specialized actions, so-called *extern objects*, which P4 does not support natively. To prevent network operators from jeopardizing the performance and line-rate capabilities of P4 switches, the language supports only a minimal set of simple operations. Loops and function calls that could be used for recursion or multiplications are, for instance, not included. To this end, extern objects are a helpful incentive for hardware vendors to add custom functionality and use them as a unique selling point.

The P4 consortium[15] provides an exemplary software switch target. These targets are implemented using the *V1Model* switch architecture on top of the *behavior model version 2 (Bmv2)*[16]. The V1Model pipeline architecture consists of a parser and two ingress elements; one for checksum verification and another one for match-action operation. As a fourth element, the V1Model holds a *Traffic Manager*, which allows external object operations not supported by plain P4. Before the deparser, there are two further egress elements, one for another match-action operation and one for checksum update. Compared to the general architecture, as shown in Figure 4, the inner match-action pipeline becomes more complex, whereas the model neglects the second parser.

14 P4 Language Tutorial `https://p4.org/assets/P4_tutorial_01_basics.gslide.pdf`, accessed 14 February 2019.
15 P4 Language Consortium, `https://p4.org`, accessed 15 February 2019.
16 P4 software switch "behavior model" version 2 ("bmv2"), `https://github.com/p4lang/behavioral-model`, accessed 15 February 2019.

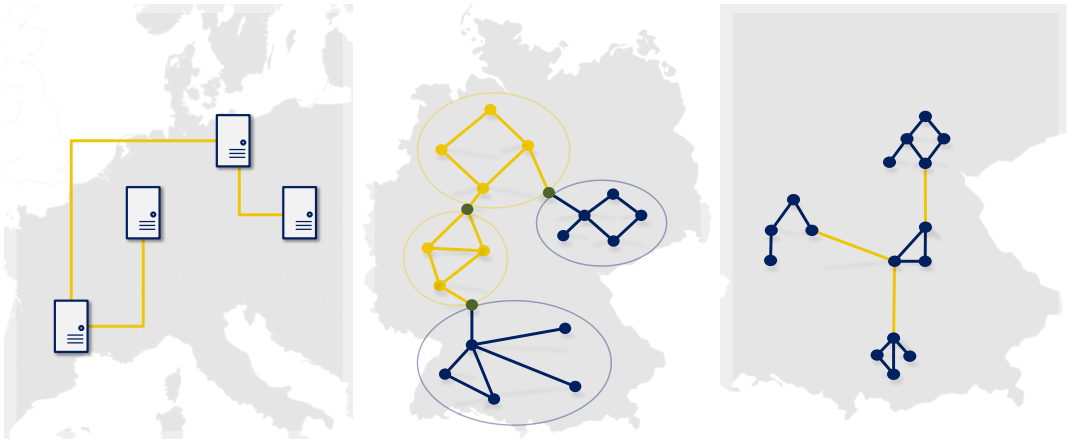
As P4 defines only the behavior of the switch, a protocol for the communication with the controller as defined by OpenFlow is still required. Dependent on the switch implementation, any protocol, including OpenFlow, can be used. The P4 consortium provides an exemplary protocol denoted *P4Runtime*[17].

2.2 Federated Software-Defined Networks

In the previous section, we introduced distributed control-planes. In addition to the physical distribution of a single network's control-plane, the majority of networks are distributed compositions of federated subnetworks. One prime reason for this is the physical distance between subnetworks. Traditionally, these networks use distributed protocols to share high-level information, e.g., BGP [RLH06]. However, in fully-fledged SDN scenarios[18], distributed SDN control-planes are inevitable.

2.2.1 *Use Cases*

In the following, we show three exemplary federated network use cases, which are sketched in Figure 5.



(a) Distributed data centers connect a large number of small data centers in any location in the world.

(b) The Internet is a network of autonomous networks interconnected through Internet Exchange Points.

(c) Distributed enterprise/campus networks are smaller interconnected department networks.

Figure 5: Three exemplary use cases, including a distributed data center, Internet ASs, and distributed enterprise/campus networks, for large-scale networks consisting of multiple federated subnetworks.

17 P4Runtime Specification `https://github.com/p4lang/p4runtime`, accessed 15 February 2019.

18 Google's B4 project [Jai+13] and Microsoft Azure data-centers [Gre15] are paramount examples for successful large-scale SDNs.

Distributed Data Centers: New applications, such as 5G, digital medical care, and automotive services arose recently. They impose challenges on the existing infrastructures demanding requirements such as latencies in the range of milliseconds and bandwidths of multiple gigabits. To fulfill these requirements, it becomes necessary to place data centers closer to the end users. Thus, data centers nowadays are distributed over numerous locations as sketched in Figure 5a. Amazon's Web Services (AWS) global infrastructure currently maintains 58 regions worldwide and plans four additional[19]. Another example is Microsoft's Azure data center network that consists of 42 regions worldwide with another 12 planned in the moment[20]. Despite their distributed locations, the data centers are federated and collaborate with each other. As a considerable portion of traffic traverses between data center [Roy+15], e.g., to synchronize state, there is a strong need to manage federated data centers collaboratively.

Internet autonomous systems: The Internet consists of a tremendous number of devices and connections organized and operated by different Internet Service Providers (ISP), such as AT&T and Deutsche Telekom [Gao01]. ISPs also organize their network in multiple autonomous systems (ASs). Other organizations, such as universities and companies, contribute with their own, dedicated ASs. An AS connects to other ASs through Internet exchange points (IXPs). Figure 5b depicts this structure. ASs of different ISPs may also overlap in locality. However, ASs are interlinked networks that set up federated networks together with other ASs from the same ISP. As we, in this work, restrict to networks that are willing to cooperate, we do not consider ASs from different ISPs as a federated network, although they predominantly carry each other's traffic.

Enterprise/Campus Networks: Enterprise networks and campus networks are federations of multiple small to medium sized networks in different locations, as sketched in Figure 5c. Depending on the size of the corresponding company or university, they can be co-located within ranges of a few kilometers. Most enterprises and universities require users to connect via VPNs to their internal network. As in most cases all participants must be able to access shared resources, the small subnetworks federate to one large, distributed network.

19 Amazon Web Services Global Infrastructure `https://aws.amazon.com/de/about-aws/global-infrastructure`, accessed 22 February 2019.

20 Microsoft Azure Regions `https://azure.microsoft.com/en-us/global-infrastructure/regions`, accessed 22 February 2019.

2.2.2 *Distributed Control-Plane Approaches*

In 2012, IETF[21] published a work in progress draft for an *east-west interface* communication standard between controllers, denoted SDNi [Yin+12]. The draft describes a protocol that is mainly used to synchronize reachability, forwarding, and capabilities among controllers of different networks. The standard assumes that all participating controllers are under the control of a single operator or willed to collaborate. However, although SDNi has not been published as standard, OpenDaylight adopted it as ODL SDNi project[22]. DISCO [PBL14], proposed by Phemius et al., follows a comparable target. The system focuses on the communication between controllers while they manage their own part of the network. To this end, a number of agents based on AMQP[23], each with a distinct purpose, set up a system to support end-to-end network services. One of the agents monitors inter-controller connections to detect failures on links that peer two subnetworks.

In addition to works on inter-controller communication, many works propose sophisticated approaches to design distributed control-planes. A pioneer work here is HyperFlow [TG10], proposed by Tootoonchian et al. in 2010. This work proposes to use local controllers to retain low response times towards the data-plane and passively synchronizing the network view-wide on controllers using publish/subscribe mechanisms. Kandoo [HG12] is an approach that splits the control-plane into multiple layers. The lowest layer is directly responsible for all events that do not require information from or reactions in other network parts. The authors assume that those events occur the most. Complex events are, in contrast, forwarded to the higher layer controller, denoted *root controller*, and processed with the network-wide view available. The approach allows implementing the root controller recursively using multiple layers. The ONOS [Ber+14a] system is a sophisticated platform to accommodate multiple controllers of different network parts into a superior network operation system. It provides support to share a global network-wide view; however, it can also be configured to favor performance over consistency. Thereby, Berde et al. discuss the performance versus consistency trade-off in detail and define requirements for large-scale operation. Similarly, a prominent work by Koponen et al., namely ONIX [Kop+10], allows flexible configuration of the distributed control-plane with respect to consistency, scalability, and persistence. An ONIX control-plane directly operates on the network-wide view while the system provides the functionality to synchronize this view between physical controllers. ONIX provides a distributed database for durable network state information and a distributed hash table for volatile information. All mentioned works so far face a trade-off between

21 Internet Engineering Task Force https://www.ietf.org, accessed 18 Feb. 2019.

22 OpenDaylight Documentation https://docs.opendaylight.org/en/stable-fluorine/developer-guide/odl-sdni-developer-guide.html#overview, accessed 18 February 2019

23 Advanced Message Queuing Protocol (AMQP) is a message broker for middlewares https://www.amqp.org, accessed 21 February 2019

consistency and performance regulated by the state distribution mechanisms. In 2012, Levin et al. discussed this trade-off [Lev+12].

Other than distributed control-plane architectures and systems, several works pay attention to placement and provisioning of controllers. Leading that field, Heller et al. [HSM12] discuss the problem of the placement and number of controllers with respect to the latency to the data-plane. Interestingly, they found that single controllers are sufficient for many investigated topologies. Furthermore, Ros et al. [RR16] similarly investigate where and how many controllers are required as well as the assignment of switches to controllers. This work particularly focuses on a fault tolerant, thus reliable operation. Lange et al. develop a software [Lan+15] to generate a Pareto-optimal placement of controllers. As the calculation of this placement lacks efficiency and thus is not usable in an online manner for dynamically changing networks, the software includes heuristic methods to approximate optimal placements. Intermeshed with placement are works that specifically investigate the dynamic growth and shrinking of control-planes. ElastiCon [Dix+14], Pratyaastha [KCG14], and Medieval [SSC15] propose elastic control-plane architectures that allow dynamically adding and removing controllers. Bari et al. [Bar+13] propose the same while minimizing the latency to the control-plane, thus also considering placement and switch-to-controller assignment. Wang et al. [Wan+16] target the same while also considering response times.

Besides general approaches to design distributed control-planes, some works propose concepts for distributed applications on top of distributed control-planes. Tam et al. [TC11] propose an architecture, where controllers are responsible for their network part only; however, they synchronize to enable full-network coverage. In contrast to other works on that, they use the example of flow allocation mainly to demonstrate a mechanism for distributed load-balancing. Furthermore, Phan et al. [PTK13] propose a model for collaborative management of multi-domain SDNs. They demonstrate the use of their model using a distributed routing mechanism. The model is based on a central collaboration entity, comparable to the two-tier management model developed by Terzi et al. to coordinate domain controllers using a central broker for legacy networks in 1999 [Ter+99]. Worth noting is also SHEAR [MS15], an interesting failure detection and resolving mechanism that combines a physically distributed control-plane with a logically centralized one. In this work, the distributed control-plane controllers localize failures, whereas the failure resolution is left for the centralized component with a network-wide view.

Summarized, the field of distributed control-planes is well studied and a decent number of sophisticated approaches exists. These works build one foundation of this thesis, as monitoring mechanisms in distributed control-planes require reliable and performant information exchange among controllers. Despite the architectural base, works shown so far do not consider data-plane

monitoring as part of the control-plane architecture. Solely, the mentioned DISCO approach monitors connections between different network parts.

2.2.3 *Isolated In-band Control-Plane Communication*

Distributed control-plane approaches provide the basis for collaborative network management among multiple controllers. The previous section listed existing comprehensive approaches, supplying communication channels, mechanisms for distributed state synchronization, fault tolerance, and much more. However, in some cases, particularly, for the development and testing of distributed algorithms, there is no need for an all-embracing solution but rather for simple communication among controllers. To this end, the authors propose a lightweight, easy-to-use, and easy-to-integrate communication system for distributed control-planes [Har+17a] without introducing as much complexity as existing approaches. The proposed system is used to establish communication among controllers in most of the contributions of this thesis. Nevertheless, it is replaceable with any distributed control-plane implementation that provides application-specific inter-controller communication. The system establishes in-band communication channels so that there is no need for a dedicated out-of-band network solely connecting controllers. In-band channels use data-plane connections to set up logical connections between controllers. Logic in-band control channels must be isolated from all other data-plane traffic to ensure security.

The proposed solution includes a discovery protocol to detect other controllers in range. For this, it observes newly connected ports on each switch under the control of a controller. Following Figure 6, the controller dispatches discovery messages through newly detected links. If another controller receives such a discovery message, it activates the connection using the port from where it received the discovery message. In addition, the controllers forward the discovery messages to all earlier discovered controllers. If a controller creates a control path using a forwarded discovery message, all controllers in between set up intermediate paths connecting both controllers and recursively active the path towards the discovery message source. [Har+17a] contains a detailed description of this sequence.

While establishing channels between controllers, the system ensures isolation from other traffic. This is achieved using VLAN- and port-based filtering, as shown in Figure 7. Switches handle only packets carrying a reserved VLAN-ID, i.e., `1002`, as control channel messages. However, the system configures switches to drop all packet with this VLAN-ID if it does not specifically expect the ingress ports and addresses. Accordingly, switches drop all packets from unexpected sources. Using One-Hot Coding [HH10], the system effectively convolves forwarding rules for different controller-to-controller channels on similar path's. A proof of concept is available in the paper [Har+17a] and a prototypical implementation is available online [Har17].

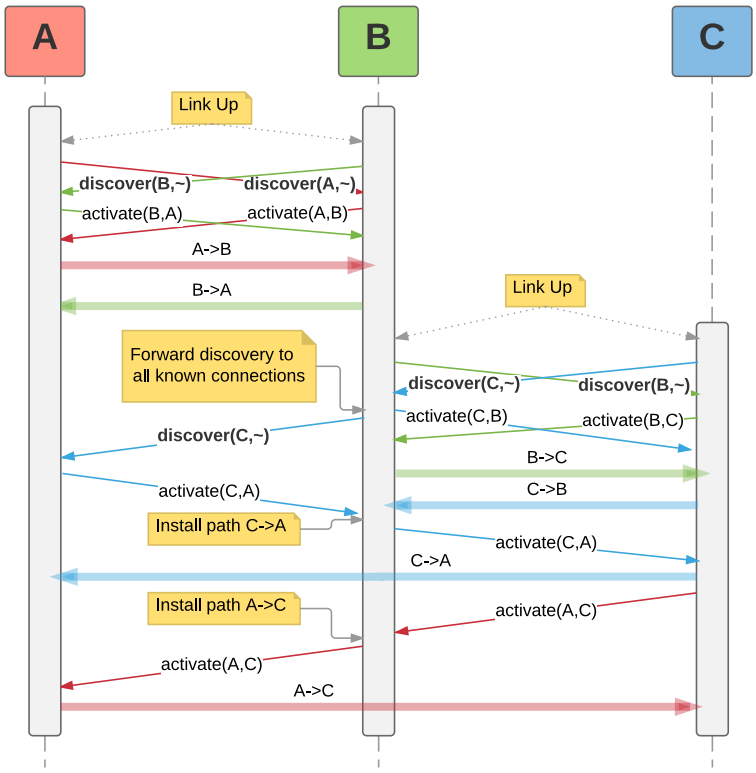


Figure 6: The detection of a new link triggers discovery messages through this link. Receiving a discovery message triggers the setup of a new control channel through the new link. Discovery packets are forwarded to already discovered controllers recursively [Har+17a].

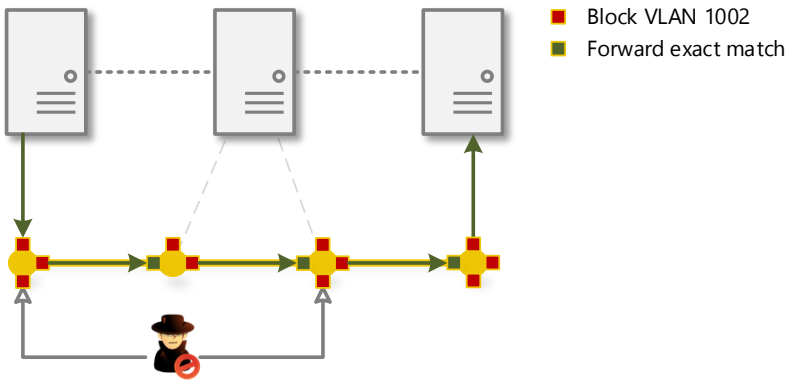


Figure 7: The in-band control channels are isolated from productive traffic using port-based and match-based filtering so that no unauthorized entity can access the channel.

2.3 Monitoring in Communication Networks

Monitoring is one of the cornerstones of network management and lays the foundations of adaptive networking. To monitor a communication network means to observe its physical and logical entities. Physical entities are networking nodes such as routers, switches, and connections, whereas logical entities include end-to-end connections, processes [Joy+87; MS93], etc. The observed information serves as data basis for the capability of the management to react to the networks state. Modern, adaptive networks continuously monitor the network state and change according to observed or foreseen changes and demands [KR12]. In addition, the control-plane itself, including management and monitoring functionalities, is monitored [Zha+16] and subject to adaptivity [Gro+13], which we also elaborate in this thesis.

The monitoring process involves roughly the following four steps, while the order or the completeness is not compulsory [Joy+87; MS93; KR12].

Measurement: First and foremost, monitoring mechanisms measure statistical information. There is a plurality of measurement techniques available; however, it can be discerned between active [AMM98; Bac+07; DR02] and passive measurements [BV02; Fri+09; Pap+06]. Active measurements either influence the entity under test to compare the behavior with previously seen behavior or use probes to check how the entity processes them. As an example, to actively measure the latency of a link, a probe packet is dispatched at the ingress of the link and captured at the egress. Timestamp differences allow computing the latency. Passive measurements use the state of an entity as basis. Measuring the used bandwidth of a link can be done passively by counting the bytes floating.

Data Collection: After conducting measurements, monitoring sinks collect the corresponding statistical data. Again, a variety of possible collection techniques exists. A possible classification dividing them is *push*- and *pull*-based. Typically, a central monitoring sink explicitly requests statistics (pull) from a measurement unit. This is particularly often the case in centrally controlled networks, such as SDNs, where the controller is widely used as monitoring sink. Nevertheless, some measurement units actively push statistical information to the sink.

Occasionally, before, during, and at the end of the data collection procedure, different preprocessing steps such as *(i)* sampling, *(ii)* aggregation, *(iii)* filtering, and *(iv)* derivation can be found. *Sampling* is a technique to take a small number of samples instead of all possible statistics in order to avoid overwhelming information floods [DLT05; SG13]. Furthermore, *aggregation* helps to get large amounts of statistical information manageable. Aggregation generates coarse-grained information from many pieces of information [Zha13]. Also *filtering* allows reducing a large amount of information to a manageable amount. However, filtering im-

plies to sample with respect to content [Cho+14b; Cho+14a; vDK14]. In all three cases, there is a danger of losing information [SRZ10]. In contrast, *derivation* is a possibility to gain additional abstract information based on primitive metrics. As an example, calculating a bandwidth from succeeding byte counter values in combination with time duration is a simple method to derive the bandwidth from primitive counters.

Analysis: Through the analysis of information, the measurements gain meaning. In this step, the monitoring unit interprets measured primitive and derived statistical information [Chi+15b; Loc11]. For example, bandwidth measurements can be compared to historical measurements and resource constraints to detect the risk of congestion.

Data Dissemination: Depending on the type of the network structure, there is need for monitoring data dissemination. Particularly in highly distributed networks, such as mobile ad hoc network, monitors again disseminate information to other endpoints [Ric+16a; Ric+15a; Bur+14]. In SDNs, monitors potentially forward measured information to management applications that use the information to adapt the networks state.

In legacy networks, it is common practice to use monitoring tools, such as SNMP [WHP99], sFlow [PPM01], or IPFIX/NetFlow [Cla04], to measure and collect statistical information from network devices. In general, they utilize agents at the devices that measure configurable metrics. In addition, a central collector gathers the measured information and provides it to management applications. With emerging SDN implementations, such as OpenFlow, those tools fade from the spotlight. OpenFlow provides flow- and port-level counter, failure information, device configurations, etc. to the centralized controller, thus, replacing a large fraction of features provided by SNMP and the alike. However, due to the fixed set of metrics retrievable using OpenFlow, enriching statistical information using legacy monitoring tools is reasonable, as Giotis et al. demonstrate for anomaly detection and mitigation [Gio+14].

The following sections describe monitoring approaches related to the work in hand. Mainly, but not exclusively, they focus on works relevant for the motivated distributed SDN network scenario described in Section 2.2.

2.3.1 *Monitoring Approaches Designed for SDNs*

The list of monitoring approaches designed for legacy networks is inexhaustible [Bol93; PCD03; Pax97; YKT99; ZD01]. Moreover, the number of approaches particularly designed for SDNs is enormous and steadily growing. This subsection points out popular approaches and directions in the latter field. The subsequent two subsections, furthermore, limit their scope to the motivated scenario and works especially relevant to the work in hand. Further, we show open gaps in the state-of-the-art that this thesis tackles.

Considering SDN, most works use passive measurements or samples collected at the switches using protocols such as OpenFlow [SOS13]. However, some works leverage the possibility to probe the network actively. SLAM [Yu+15], proposed by Yu et al., is an active approach exploiting SDN control mechanisms to measure the latency of any path in the network. To this end, SLAM dispatches specific probes that trigger control messages on the first and the last switch of the investigated path. Atary and Bremler-Barr propose GRAMI [AB16], also measuring latencies, precisely, round trip times of all links and paths between all nodes. To do so, the authors place vantage points according to their needs. The decision, whether active probing or passive measuring is of choice, has been discussed earlier by Barford and Sommers [BS04].

Passive measurement approaches do not actively disrupt productive processes in the network. The vast majority of works designed to monitor Software-Defined Networks make use of counter values gathered at the switches to infer desired information. Generic examples for this procedure are OpenNetMon [vDK14] and the bandwidth measurement method proposed by Megyesi et al. [Meg+16]. OpenNetMon is a framework providing end-to-end Quality of Service (QoS) metric. The approach optimizes the polling frequency to achieve good accuracy while maintaining the CPU resource consumption of switches. Megyesi et al. limit their measurements to the bandwidth metric; yet, develop a reasoned method to provide measurements for all node pairs in the network by considering the whole topology. Bozakov et al. [Boz+16] propose to randomly alter the counter polling frequency to provide a more trustworthy view on characteristical flow statistics. Yu et al. propose FlowSense [Yu+13] an exclusive method to passively measure the bandwidth without generating additional costs. The authors propose to leverage OpenFlow control messages dispatched from switches on an occurrence of new flows and their eviction. The method infers network utilization using statistical information of flows within the control messages. While the authors propose a peerless method with zero overhead, the method suffers in timeliness, particularly facing long-term flows. In addition, the method is strictly limited to reactive routing networks as they rely on OpenFlow's *Packet-In* and *Flow Timeout* control messages.

Sampling, introduced earlier in this chapter, is a useful passive method to reduce large amounts of information into smaller chunks of representative information. SDN implementations, such as OpenFlow, provide handy mechanisms for traffic mirroring but not sampling only parts of a flow. To overcome this shortage, Shirali-Shahreza and Ganjali propose FleXam [SG13] to sample packets on flow-level as an extension for OpenFlow. As an alternative, MonSamp [RSC14] utilizes a middleware to sample entire flows and redirect them to the controller if required. The proposed middleware provides generic QoS monitoring of flows while adapting the sampling rate depending on the faced bandwidth utilization. OpenSAFE [BRA10] follows a comparable ap-

proach by providing a language, ALARMS, to program devices to re-route traffic to monitoring appliances. Rather than taking part in the actual traffic analysis, the authors focus on directing traffic in line-rate through security applications. Much effort has been put into the investigation of sampling methods. As naïve packet sampling underrepresents small flows, Duffield et al. [DLT05] attempt to solve this by sampling packets based on their type, i.e., sampling only TCP SYN packets. A further packet sampling approach allows estimating flow lengths, leveraging TCP sequence numbers of sampled packets [Rib+06]. In contrast to packet sampling, flow sampling is becoming more important in the SDN era. Flow sampling, as proposed by Hohn and Veitch [HV06], is a feasible alternative given the capabilities of OpenFlow. Tune and Veitch [TV11] discusses the advantages and disadvantages of potential sampling techniques.

Many monitoring approaches, designed for SDNs, use the concept of adaptability to improve their efficiency. Intuitive and widely used is the adaptation of the measurement resolution, directly influencing the accuracy of the network state visible to the management applications. With respect to SDNs, the resolution of measurements is twofold: *(i)* spatial and *(ii)* temporal. *Spatial resolution* describes the granularity regarding the covered flows with a single measurement. Jose et al. [JYR11] propose a detection method of heavy hitters by adaptively zooming in the IP range of interest. If an IP range contains more than a configured portion of the traffic, the method increases the granularity for this range ("flow rule expansion"). Figure 8 explains the rule expansion, which zooms into the area of interest.

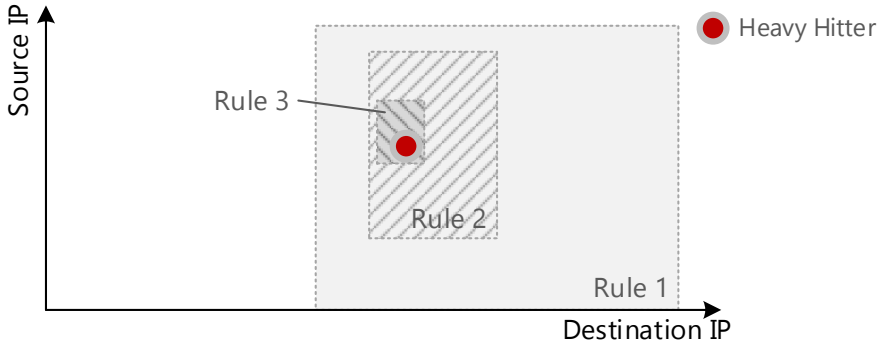


Figure 8: Flow space represented with two header fields: Source IP and Destination IP. *Rule 1* covers a large portion of the flows. Once the method detects a massive increase of traffic, it uses *Rule 2* to zoom into the portion of interest and, within that, finally into an even more fine granular portion of the traffic with *Rule 3* to identify the heavy hitter.

In addition, OpenWatch [Zha13], an anomaly detection method using flow counters, adapts the flow rule granularity to zoom into rules that aggregate multiple sub-flows. The work additionally introduces the *temporal resolution* of measurements. The author increases the frequency of measurements for

rules of special interest and decreases it for unsuspicious ranges to reduce costs. OpenNetMon [vDK14], also mentioned earlier, is an exemplary work adapting the temporal dimension of periodic measurements. The approach increases and decreases the polling rate analog to OpenWatch to maintain the accuracy with reasonable CPU load. PayLess [Cho+14c], proposed by Chowdhury et al., follows the same adaptation approach. A slightly different approach to adapt the monitoring is proposed by Tangari et al. [Tan+17a], who change completely between two monitoring configurations. The default "light" configuration solely collects packet and byte counter. Whereas the "heavy" configuration, used for flows of particular interest, additionally analyzes TCP information in depth. The authors argue to reduce the CPU utilization per packet while maintaining a reasonable accuracy of their metric of choice.

Numerous applications have been developed leveraging SDN mechanisms. As an example, a typical monitoring application is anomaly detection. Afek et al. [Afe+15] describe the detection of elephant flows under the assumption that future SDN protocols support sampling. Furthermore, NetFuse [Wan+13] detects aggressive flows that overburden the network using solely the passive OpenFlow counter collection mechanism and shapes the traffic of such flows rather than reconfiguring the network. To stick to the application example of anomaly detection, we find a number of related approaches for SDNs. Hand et al. [HTK13] use SDN mechanisms to add programmability to the security feedback loop. Their integrated approach embraces sensing the infrastructure, adjusting the configuration on the fly, and collecting forensic information. The L-IDS [SBB13] system shows how to utilize SDNs to support intrusion detection and correspondingly reconfigure the network. Other works, such as OrchSec [Zaa+14] and EVOLVE [Tan+14], provide the basis to support the development of security applications. Scott-Hayward et al. [SNS16] provide a broad survey about security-related approaches in the context of SDN.

So far, this section described generic SDN monitoring frameworks, active and passive approaches with different sampling schemes, self-adapting monitoring solutions, and anomaly detection mechanisms as exemplary application. Since it is essential for the work in hand, we further discuss existing approaches focusing on the placement of monitors and measurement points as well as the integration of monitoring frameworks into the SDN architecture.

Considering the placement of monitors and measurement points, many aspects of approaches designed for legacy networks [Can+06; Suh+06; Cha+05; Dol+09] can be reused in SDN scenarios as well. Particularly considering SDNs, Tootoonchian et al. [TGG10] proposed strategies to select measurement locations to capture byte counter using OpenFlow: Within this work, the system denoted OpenTM uses different criteria such as the last switch on a flow's path, a random switch on the flow's path, round-robin selection among switches, etc. They evaluate the different criteria against their achieved accuracy when using the measurements to fill a traffic matrix [Tun+13; Med+02]. The authors propose a simple yet effective method to generate a fully mea-

sured ingress-to-egress traffic matrix. Furthermore, Yoon et al. [Yoo+17] make use of centrality measures [Fre78] to estimate the relative importance of nodes in the network to optimize information collection for an Intrusion Detection System (IDS). The work refers to an important aspect when using centrality measures for communication networks[24]: Traffic might have different characteristics in different regions. The core of the network is tendentially emphasized using centrality measures but faces different traffic types than edge regions. However, centrality measures reflect only the physical structure of the data-plane.

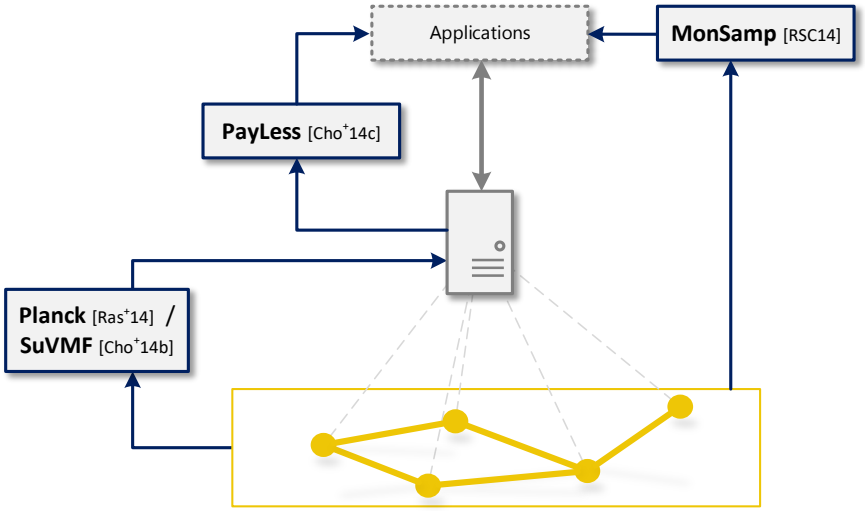


Figure 9: Integration of proposed monitoring frameworks in the SDN architecture: PLANCK [Ras+14] and SUMA [Cho+14b]/SUVMF [Cho+14a] integrate as an intermediate layer between the control layer and infrastructure layer (yellow). PAYLESS [Cho+14c] is an intermediator between the control and application layers. MONSAMP [RSC14] enriches monitoring information given from the controllers with sampling-based information directly captured from the infrastructure.

Many of the aforementioned approaches either integrate as applications in the SDN architecture or optimize the statistic collection process between controller and the data-plane switches. Nevertheless, as sketched in Figure 9, works such as PLANCK [Ras+14] explicitly utilize a new entity in the control layer. PLANCK uses line-rate samples to buffer statistical information and provides metrics, such as flow rates and link utilization, to the upper layer, i.e., controllers. Similar to that, SUMA [Cho+14b] and the follow-up SUVMF [Cho+14a] approaches introduce a middlebox acting as a proxy for statistical information. It provides filtering and abstraction for monitoring as well as basic functionality for an event-based information stream between

24 Centrality measures originate the social network theory [Fre78].

the data-plane and controllers. MONSAMP [RSC14] proposes a entirely orthogonal approach, with an additional element on top of the control-plane. Controllers connect to the MONSAMP entity via their northbound interfaces. MONSAMP utilizes mirrored traffic to enrich the information collectible in the controller using OpenFlow. Given that, the approach provides a variety of monitoring information tailored for traffic engineering. Chowdhury et al. propose PAYLESS [Cho+14c], a monitoring framework between controllers and applications. Its main contribution is the abstraction for applications: it translates low-level information in meaningful information and relieves applications from dealing with selecting suitable measurement points and the like. Giotis et al. [Gio+14] do not explicitly mention additional management entities but propose an architecture to combine OpenFlow statistics gathered in a controller with sFlow statistics in a logically separate collector for anomaly detection and mitigation modules. In the implementation, the authors; however,combine both in the same controller.

Gong et al. [Gon+15] broadly survey works focusing on Software-Defined Networking applications, including monitoring. The works highlighted so far are regardless of the distributed nature of federated networks as motivated in Section 2.2. Therefore, the next subsection depicts approaches particularly considering the distributed architecture of SDNs.

2.3.2 *Monitoring Approaches Designed for Distributed SDNs*

Forestalling the takeaway of this subsection, we find a lack of monitoring mechanisms specifically designed for federations of networks that are controlled by separate controllers. The following describes collaborative approaches as well as approaches, which consider the distributed nature. Decentral, collaborative monitoring approaches designed for legacy networks without central control [SA12; PS06] cannot be applied in Software-Defined Networks right away. First, such approaches are unaware of established measurement techniques like counters provided by SDN. Also, the dominant role of controllers that leverage their superordinate network-wide view and control change the prerequisites fundamentally. In SDN scenarios, it remains clear that not even powerful single physical controllers can handle the load when the complete network state is monitored [KCG14; HG12; Gio+14].

The CSAMP [Sek+08] system proposes network-wide flow monitoring. Instead of "network-wide" with respect to the control-plane, the approach targets towards coordination of the routers within one network-part. The system distributes flow sampling responsibilities among routers of a network without explicit communication. Comparable to that, DCM [YQL14], "Distributed Collaborative Monitoring" for SDNs, developed by Yu et al., coordinates the measurements in switches within a single network. The authors propose a method using bloom filters to eliminate redundant and overlapping flow measurements, respectively, in neighboring switches. Their approach targets to

lower memory consumption while retaining accuracy. However, it enables collaboration on the data-plane but not on the control-plane with federated controllers. Towards the data-plane coordination, UNIVMON [Liu+15; Liu+16] provides a framework for a "one-big-switch" abstraction in SDNs supporting network-wide monitoring. The OPENWATCH framework [Zha13], shown previously for its adaptive flow rule expansion function, includes functionality for network-wide placement of monitoring rules. Furthermore, OPENTM [TGG10], mentioned earlier, provides switch selection within a network. A-GAP [PS06], by Prieto and Stadler, is a system to join network-wide monitoring information through a tree towards the root, which is the controller.

Kozat et al. [Koz+16] seize another interesting aspect by considering control-plane partitioning. Their work for failure localization offloads controller functions by pushing as much functionality onto the data-plane as possible. The work proposed by Chin et al. [Chi+15a] contains an approach for collaborative monitors that conduct anomaly detection. The approach correlates traffic observed at different monitors through the centralized controller. While this work proposes a distributed, collaborative analysis application, the focus of this thesis is collaborative measurement of the network state, which is the basis for any analysis. Taheri Monfared and Rong [TR13] introduce an interesting approach to discern monitoring information for different tenants in an Infrastructure-as-a-Service environment. The authors argue that current approaches are unable to differentiate between involved parties. By analogy to previously described works, the multi-tenant monitoring takes the distributed infrastructure of the underlying data-plane into account rather than the distributed nature of networks as a whole with federated control.

Solely Tangari et al. [Tan+16; Tan+17b] propose a distributed approach to monitor large-scale networks consisting of multiple domains in the context of SDN. In their approach, they propose a monitoring coordination system supporting a wide range of measurement tasks with various granularities. The system is based on interconnected, yet autonomous monitoring agents adopting the architecture proposed in the distributed control framework from [Tun+15]. The approach from Tangari et al. provides the functionality to delegate monitoring tasks to, and share information between different parts of the network. However, the system does not take advantage of collaboration of the federation of networks.

2.3.3 *Using Programmable Data-Planes for Monitoring*

With the limitations of OpenFlow and the resulting advent of P4 [Bos+14], programmable data-plane elements also came into view to support monitoring. Even before P4, several works with great potential to enhance the data collection process were proposed. With the aspect of dynamically programming the devices, making them independent of custom hardware, their actual deployment is within reach.

Already in advance of SDN living up, Bandi et al. [Ban+07] propose specialized network processing units (NPU) in combination with TCAM within switches to boost up the detection of heavy hitters. DEVOFLOW [Cur+11] proposes intelligent switches that replicate rules to allow autonomous "flow rule expansion" as known from adaptive works described earlier [JYR11; Zha13]. The switches increase the granularity for critical flows so that they are able to inspect mice flows and detect elephant flows without involving the controller. The "Distributed Collaborative Monitoring" (DCM) [YQL14] work uses Bloom filters[25] in the switches to select the subset of flows to be monitored. Their proposed method leverages Bloom filters to correlate measurement rules in a way, that dissects overlapping rules to provide a fine-granular flow statistics avoiding a vast number of fine-granular rules. Drawing on Bloom filters, sketches[26] provide an excellent opportunity to improve monitoring efficiency in terms of memory consumption in switches. For example, OPENSKETCH [YJM13], proposed by Yu et al., is a solution utilizing sketches to support customizable data collection. Using measurement libraries on the control-plane, controllers can configure the measurement pipeline in the switches according to the required metrics in a handy way. Figure 10 illustrates the principle of a sketch, in particular, a *count-min sketch*.

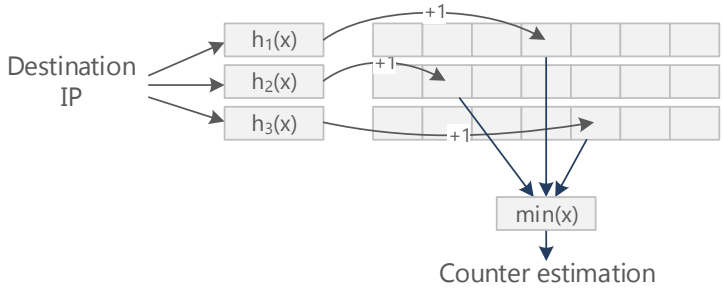


Figure 10: The count-min sketch hashes a header field (here destination IP) using multiple functions h_1, h_2, h_3 to generate the indices. The counters at the corresponding indices are incremented. To fetch the counter value for this destination IP, the same hash functions are used to identify the counters indices. The minimum of these counters is used to estimate the true counter value. As the counter size can be selected much smaller than using an entry for each destination IP, the memory consumption is lowered. The count-min sketch gives an upper bound of the counter.

Prominent works using sketches are also SCREAM [Mos+15] and its predecessor DREAM [Mos+14]. The proposed switches enable the collection of statistical information with respect to a user-defined accuracy. Using sketches, SCREAM stores the collected data, in analogy to OPENSKETCH, optimized in

25 Bloom filters are efficient data structures to check if elements are in a set. False positives are possible but no false negatives [BM04].

26 Sketches are efficient data structures to summarize element counts in a set. Sketches increment array entries based on the element's hash [YJM13].

limited SRAM. Both, OpenSketch and SCREAM, have to specify the targeted metric, that is potentially firmly application-specific, a-priori such that they do not track any other metric automatically. Yang et al. [Yan+18] propose elastic sketches that adapt to the traffic characteristics in order to quickly monitor with high accuracy. Huang et al. [HLB18] use sketches on the data- and control-plane to optimize the trade-off between resource usage and accuracy.

Furthermore, Liu et al. propose UnivMon [Liu+15; Liu+16], where switches collect generic sketches that the control-plane uses to generate application-specific metrics. With their approach, the authors target generality and fidelity at the same time. The opportunity to program switches opens up the potential to detect heavy hitters effectively without involving the control-plane as Popescu et al. [PAM17] and Sivaraman et al. [Siv+17] show. Despite monitoring applications, programmable data-planes have tremendous potential to support any management applications [Kun+18; Li+16; Siv+16].

2.4 Summary

In this chapter, we introduced Softwarized Networks, which build the foundation and context of this work. Besides background information on SDN, OpenFlow, NFV, and programmable switches, the chapter motivated and justified the scenario of this work, namely federations of Softwarized Networks. On top of this, a survey on existing monitoring works that use these new technologies was given. In addition to a general overview, we identified a lack of monitoring solutions particular with respect to the motivated scenario, i.e., collaborating, federated networks.

3

Collaborative Network State Monitoring

Software-Defined Networks consist of distributed control-planes to ensure responsiveness and scalability as well as to prevent fail-safety issues [TG10; HG12; Ber+14a; Kop+10; Lev+12] (cf. Section 2.2.2). Furthermore, the majority of today's large-scale networks is organized in a distributed fashion, compiling a federation of subnetworks (cf. Section 2.2.1). Concerning coexisting centrally controlled networks, either single networks with a distributed control-plane or as a federation of subnetworks, we found a lack of mechanisms for collaborative network monitoring. As a consequence, currently, subnetworks do not cooperate while collecting statistical information. It has repeatedly proven that naïve monitoring approaches cannot sufficiently handle the required monitoring demands to oversee the overall network state [Gio+14; HG12; KCG14]. Moreover, mechanisms such as aggregation, sampling, and filtering distort the real state of the networks. To overcome this shortage, the first major contribution of this thesis is the development of *cooperative approaches to coordinate monitoring within collaborating softwarized networks*. In this thesis, collaborative monitoring among multiple networks focuses on the collection of monitoring information that is of interest for multiple network parts. This includes the inference of statistical information based on measurements conducted in neighboring parts of the network.

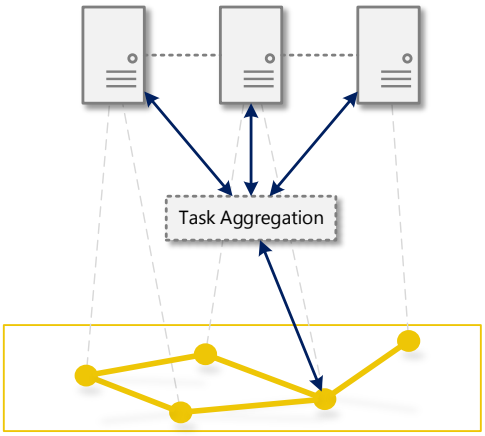


Figure 11: Collaborative monitoring using a centralized entity that correlates monitoring interests of different controllers.

To enable coordinated monitoring in federated SDNs, we develop, in Section 3.1, an architecture that leverages a centralized coordinator as sketched in Figure 11. This entity gathers information about monitoring tasks from cooperative controllers. With full knowledge of monitoring interests, it convolves and aggregates monitoring tasks (cf. [CL00; LC06]) aiming to increase the monitoring efficiency while serving the controllers with all required information to obtain their network state.

In addition, in Section 3.2, we extend the coordination approach towards a robust design that does not involve additional entities, as depicted in Figure 12. That approach only coordinates controllers and does explicitly not participate in the information collection process.

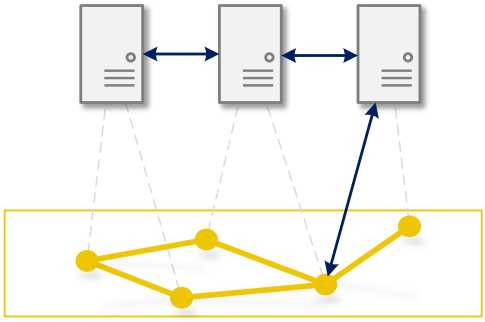


Figure 12: Collaborative monitoring among network parts while the information collection and distribution tasks remain in the controllers.

On top of showing the development and evolution of collaborative monitoring designs, Section 3.3 contains the evaluation of the approaches, which we propose in the following, with respect to their efficiency. More precisely, this refers to the monitoring cost reduction, coordination overhead, load fairness, and monitoring performance.

3.1 Centralized Monitoring Task Aggregation

The design described in this section is based on parts of our work [Har+18a]. As described and sketched with Figure 11, the first approach to coordinate monitoring tasks among multiple networks is to devolve tasks to a dedicated entity. This centralized entity, called coordinator, holds information on periodic monitoring tasks from each controller and uses this information to correlate the collection of tasks. Through correlation, multiple tasks with overlapping specifications can be convolved into a single task in order to reduce costs. The entity provides further functionality to optimize the event stream between the data- and control-plane, which is shown in detail in Chapter 4.

3.1.1 *Integration of the Coordinator in the SDN Architecture*

In line with the previous section, Figure 13 depicts the integration of the coordination entity, also denoted SRR (Statistic Request Relay) or only coordinator, into the SDN architecture with distributed subnetworks. The coordinator, acting as an intermediator between controllers and switches, accordingly lies between the control-plane and data-plane known from approaches such as FLOWVISOR [She+09] or OPENVIRTEX [AlS+14]. Controllers communicate with the northbound interface of the intermediating entity through a dedicated channel. The entity itself uses various protocols to communicate through its southbound interface with the data-plane switches to collect statistical information. The advantage of the centrally available collection of all monitoring interests in the coordinator comes initially with the disadvantages of a single point of failure. To counteract this, we propose further deployment modes that have been proven reasonable, e.g., in the context of NFV [CDJ17].

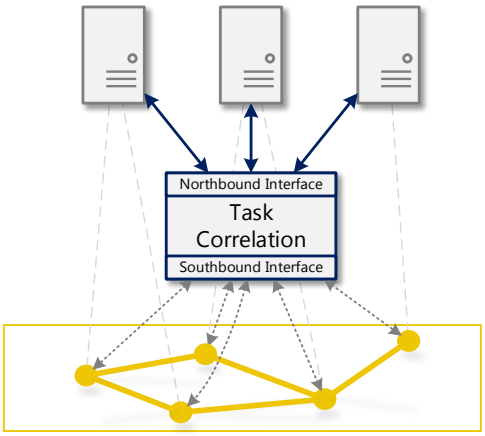
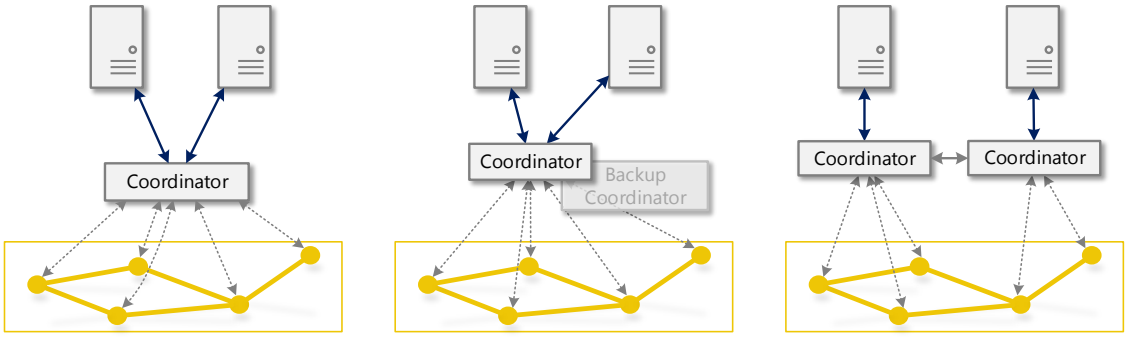


Figure 13: The coordinator is placed as an intermediator between the data- and control-plane. The controllers are connected through their southbound interface with the coordinator's northbound interface, which itself is connected through its southbound interface with the data-plane switches.

Single Server Deployment

The single server deployment mode, depicted in Figure 14a, reduces complexity and deployment costs. A single machine holds all information and does not require state exchange with other instances. The controllers, as well as the data-plane elements, only connect to one well-defined endpoint. However, a single server lacks scalability and fallback possibilities.



(a) Single Server Deployment with only one physical instance.

(b) High-availability Deployment with a backup instance that is inactive until the prime instance fails.

(c) Load-balancing Deployment uses multiple parallel instances for scalability.

Figure 14: Architecture of the different deployment modes of the logically centralized coordinator.

High-availability Cluster Deployment

By analogy to distributed control-planes, the coordinator can consist of multiple physical machines (cluster). A cluster deployment ensures high-availability by replication, i.e., as shown in Figure 14b, the centralized coordinator has a passive copy, which is not interacting with any other devices. This copy becomes active once the prime coordinator loses connectivity. By introducing such fail-safety to the system, this deployment mode also introduces provisioning and operational costs, particularly for state replication. Levin et al. [Lev+12] discuss this trade-off and show that state consistency is of significant importance and different states in such a distributed system degrade for the performance predominantly.

Load-balancing Cluster Deployment

Analogical to the high-availability deployment mode, the load-balancing deployment mode, sketched in Figure 14c, introduces multiple instances of the coordinator. However, all instances are now active at the same time, thus can be addressed through their northbound interface and communicate with the data-plane switches. Different monitoring tasks are handled by different coordinators depending on their characteristics, such as the measurement location. For small networks, the costs for the introduction of a new entity (*CAPEX*) must be evaluated against the operational cost (*OPEX*).

3.1.2 *Inner Coordinator Architecture*

Having defined the integration of the logically centralized coordinator, this section describes the design of the SRR component as such, depicted in Figure 15. We use the single-server deployment as a valid abstraction for potentially distributed implementations with state transfer or load-balancing. As the figure suggests, the component consists of three layers: The northbound interface as communication interface to controllers, the inner logic that is responsible for correlating tasks leading to coordinated monitoring and the southbound interface containing different agents to talk to the data-plane switches. Note that the boxes shown in light gray refer to functionality, which is not treated in this chapter.

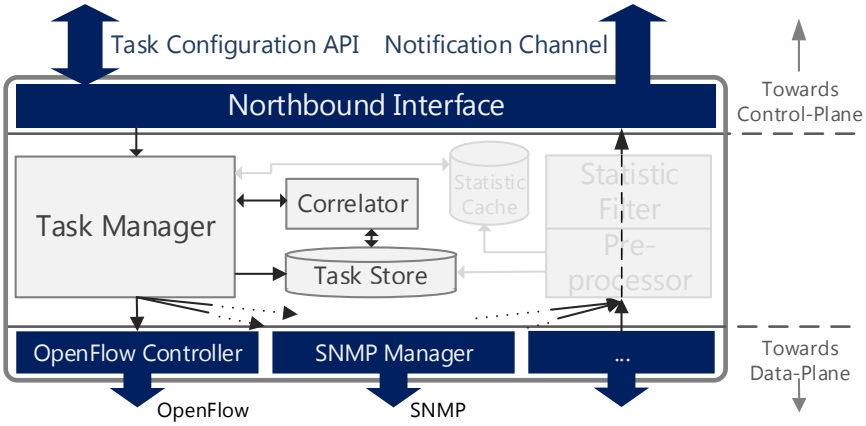


Figure 15: The coordinator is organized in three layers: the northbound interface with two communication channels towards the controllers of the control-plane; the inner logic is responsible for task aggregation; and the southbound interface pointing towards the data-plane with multiple agents speaking different protocols.

Northbound Interface

In addition to a service API, the northbound interface contains notification channels pointing towards the controllers. These channels are logically distinct due to their different purposes: The API serves as a synchronous request-response endpoint to define and configure monitoring tasks, whereas the directed notification channel delivers asynchronous event-based statistics from the coordinator to controllers.

The coordinator is equipped with a set of predefined metrics, including the information on how to derive them based on raw metrics. As an example, the coordinator can calculate the bandwidth based on two consecutive byte counter values Doing so, the analysis and derivation of a sophisticated net-

work state is left to the control applications in such a way that we minimize the space for (mis-)interpretation of information.

The API supports methods to register, configure, and invalidate monitoring tasks. This service interface uses UDP for the transmission and JSON as payload message format. A registration message contains identification information about the issuing controller and defines the full configuration of a task. The task configuration includes information about the periodicity, the metric to measure, the measurement point, and the questioned data-plane object (e.g., flow, port). In addition, the registration message specifies a degree of freedom for certain parameters. This allows correlating tasks with other tasks whose specifications overlap not entirely but are close-by. For example, a controller supports to alter the measurement point and the measurement period. We detail this in the next section (3.1.3). Listing 1 shows an exemplary task registration message.

Listing 1: Exemplary task registration message.

```
1  {
2   "controller": "10.10.10.100",
3   "controller_port": 7702,
4   "task": {
5        "entity": {
6           "dpid": 144115188075855874,
7           "port": 1
8        },
9        "metric": "bandwidth",
10       "period": 1000,
11       "periodic": true,
12       "threshold": 10000,
13       "threshold_type": "delta",
14       "aggregation_freedom": {
15           "min_period": 500,
16           "max_period": 1500,
17       }
18       "type": "register_task"
19    },
20  }
```

Line 2 and 3 identify the controller associated with the task. The remaining Lines 4–19 define the configuration of the `task`. Therein, the `entity` object, defined in Lines 5–8, points to a `port` of a switch identified by its `dpid` (*`datapath-id`*) while the subsequent Line 9 specifies that the `bandwidth` is the `metric` of interest. Hence, this task issues the measuring of the bandwidth

consumption on a switch's port, thus a link. Line 11 (`"periodic": true`) exhibits that the task is to be performed periodically with a period of 1000 (Line 10). However, in Lines 14–17, the controller allows performing the task with a higher or lower frequency (period range between `period_min: 500` and `period_max: 1500`). Lines 12 and 13 define filtering options not handled in this chapter and Line 18 identifies the message as a task registration.

Inner Coordination Logic

The architecture of SRR's logic layer is depicted in Figure 15. The major components responsible for task correlation, and thus the aggregation of monitoring tasks to reduce costs are the `Task Manager`, the `Correlator`, and the `Task Store`. The `Task Manager` receives parsed task objects and, if they are periodic, stores them in the `Task Store`. Moreover, the manager executes active tasks, either immediately or periodically. Before the `Task Manager` inserts new periodic tasks to the store, it interacts with the `Correlator`. The `Correlator` is the heart of the aggregation mechanisms and enables cooperative monitoring. It traverses the `Task Store` for similar tasks, particularly those which measure the same resource with overlapping specifications. In order to aggregate two (or more) tasks, the following requirements must be met:

- The desired *metric* (e.g., bandwidth usage in bytes/second) must be equal. It is explicitly defined in the task registration message.
- The *entity* (e.g., link between switch s_1 to switch s_2) must be equal. It is also explicitly defined during the task registration.
- The *measurement point* (e.g., *port* p_1 *of* s_1) must overlap. The applications can allow that the aggregation mechanism can use a different measurement point than specified. For example, the bandwidth on a link between s_1 and s_2 can be measured at both switches.
- The *measurement frequency* f *(update period* $\rho = 1/f$*)* must overlap (e.g., new measurement every two seconds, $f_1 = 1/2s$). The API grants the applications the opportunity to define a range of possible measurement frequencies, as, for instance, given in the `"aggregation_freedom"` of the `"task"` object in the registration, cf. Listing 1. In the example, the targeted period is 1000ms (Line 10); yet, any value between 500ms and 1500ms is acceptable (Lines 14–17). The aggregator tries to find a period that fulfills specifications of all tasks. Say controller A targets $f_A = 1/s \rightarrow \rho_B = 1s$ and controller B targets $f_B = 1/1,5s \rightarrow \rho_B = 1,5s$. Given A uses the degree of freedom as described in the listing ($f_A^{\Delta} = 1/1s \pm 1/0,5s$), the aggregation mechanism will alter f_A to $f'_A = f_A + 0,5s$ so that $f'_A \in f_A^{\Delta} \wedge f'_A \in f_B^{\Delta}$. On top of the direct overlap of the measurement frequencies, the correlation mechanism also considers tasks whose measurement period is a

divisor of another task's measurement period. The resulting period is then the smaller period:

$$\rho' = \begin{cases} \rho_B & \text{if } \exists k \in \mathbb{N}^* : \rho_B \cdot k = \rho_A \\ \rho_A & \text{if } \exists k \in \mathbb{N}^* : \rho_A \cdot k = \rho_B. \end{cases} \tag{1}$$

Thus, only if $\exists k \in \mathbb{N}^* : \rho_B \cdot k = \rho_A \vee \rho_A \cdot k = \rho_B$ (one is the divisor of the other) this method is used. For example, say $f_A = {}^1/_{2s} \rightarrow \rho_B = 2s$ and $f_B = {}^1/_{6s} \rightarrow \rho_B = 6s$. Subsequently, the frequency of the aggregating task is $f' = {}^1/_{2s} = f_A \rightarrow \rho' = 2s = \rho_A$. However, the notification channel transmits only every third measurement to controller B such that the aggregation is, in that case, transparent to the controller.

Southbound Interface

In order to request statistical measurement information from the data-plane, the coordination entity uses different agents on the southbound interface. It translates monitoring tasks to statistic requests for the data-plane switches. To do so, it uses an extensible set of agents talking different communication protocols. Since it supports using existing protocols, there is no need for custom functionality on the switches. By default, the coordinator uses a *slave* OpenFlow controller[27] agent to request information from the switches using OpenFlow. Furthermore, an SNMP agent allows measuring metrics not supported by OpenFlow. The combination of OpenFlow with SNMP supports a wide spectrum of traffic related monitoring (flow/link byte and packet counter, etc.) and capacity utilization monitoring (port failure rates, switch CPU/memory consumption, etc.), which is sufficient for most sophisticated monitoring applications. As the list is extensible, future implementations allow for agents with custom information collection mechanisms [YJM13; Mos+15; Mos+14; Liu+16] using e.g., P4 [Bos+14].

3.1.3 *Coordination Workflow*

This section elaborates the coordination workflow using the centralized coordinator starting from the task registration until the invalidation of a task. Figure 16 schematically shows this workflow. The upper lifeline depicts the controller's, the center lifeline coordinator's, and the lower lifeline switches' activity. Below the lifelines, the figure denotes the component of SRR that is predominantly interacting in the corresponding situation.

27 Controllers in SLAVE mode have read-only access to the switches. They do not receive asynchronous messages except regarding changing port conditions. They are not able to alter the state in the switch. Still, multipart requests, which include any type of statistical requests, are processed normally [Pfa+12].

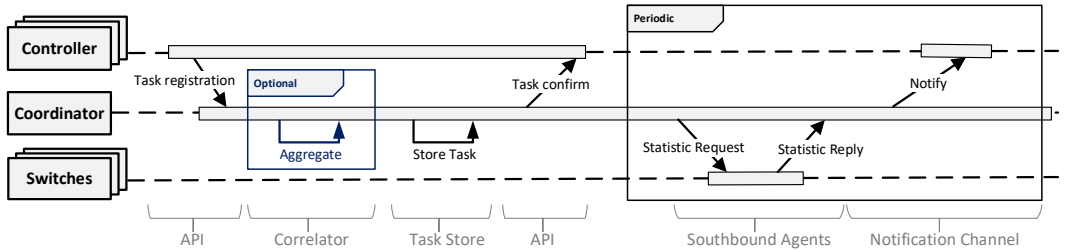


Figure 16: The workflow starts with the registration of the task, followed by the aggregation with similar, already active tasks by the `Correlator`. Subsequently, the coordinator stores new tasks and periodically executes corresponding measurements, which comprise of statistic requests, their responses from a switch, and notifying interested controllers.

First of all, within the process, a task registration dispatched from a controller reaches the `Correlator` through the API, which parses the call. Next, if possible, the `Correlator` aggregates the task with other active tasks in the `Task Store` as described in the previous section. Subsequently, it stores the task and confirms it to the controllers through the API. A task confirmation message includes the final configuration of the potentially aggregated task as it might differ from the original specification if the degree of freedom allowed, e.g., to alter the measurement frequency. Figure 17 shows the life-time of different tasks from the controllers and how they are executed or represented in the coordinator.

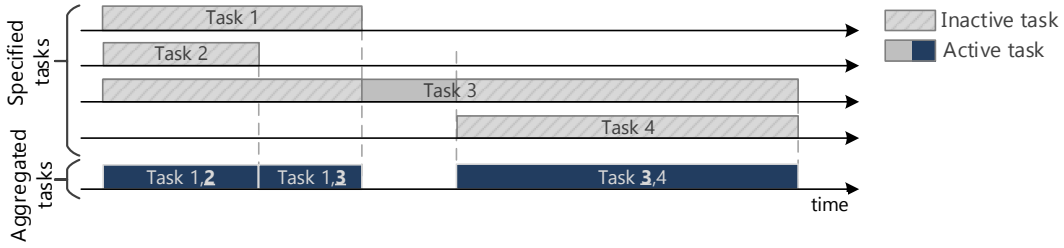


Figure 17: Exemplary time to live chart for tasks. Gray tasks are directly specified from controllers. Blue tasks are aggregated tasks. The bold, underlined number in aggregated tasks identifies the task that includes the measurements for all other tasks.

In the beginning, Tasks 1, 2, and 3 are active and aggregatable. As long as Task 2 is alive, its unaltered specification includes all measurements of Tasks 1, 2, and 3 (cf. the lowermost aggregated task timeline). Once Task 2 is removed, Task 3 can be used to represent all measurements of Tasks 1 and 3. Afterwards, when only Task 3 is active, no aggregated task is active;

however, Task 3 remains active. As soon as Task 4 appears, which again, can be aggregated with Task 3, their measurements are aggregated using Task 3.

As long as a task is active and valid, the coordinator requests statistics from the switches on behalf of the controllers (cf. Figure 17). The switches answer with the requested information used to notify controllers about updated state information.

3.2 Distributed Monitoring Task Coordination

The centralized approach introduces a single point of failure and using the proposed distributed deployment modes introduces considerable costs for state transfer. To overcome this, the following extends the collaborative approach with a robust design without an additional entity. Thereby, this section is twofold: The first subsection describes the collaborative, distributed flow monitoring approach without coordination entity but central coordination; The second subsection further extends this approach with decentralized coordination. The proposed designs enable collaboration by eliminating strongly overlapping, thus redundant measurements among multiple controllers of a federated network. In contrast to the previous section, we speak about the aggregation of measurements rather than tasks. In this work, monitoring tasks specify measurements on a high level while leaving open how the measurements are conducted. Measurements translate, in the case of OpenFlow, directly to statistic requests to obtain the information. While the approach from the previous section is generic in the sense that it is open for any type of measurement, the following approaches focus on the measurement of flows sizes. Flows are a supreme example of a shared resource among multiple federated networks. With slight alterations, the design can be adapted for other kinds of metrics, such as throughput of shared links etc. To complete the example, a monitoring application uses the flow measurements to assemble a traffic matrix for each subnetwork. A *traffic matrix* is a powerful representation of the traffic flowing between pairs of nodes in a network [Tun+13; Med+02]. The actual metric stored for each cell of the

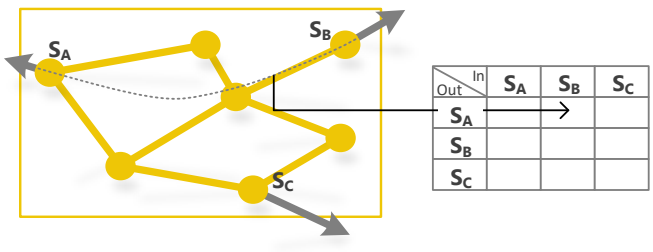


Figure 18: A traffic matrix has one cell for each ingress/egress switch pair. All traffic entering the network at switch S_A that leaves the networks through S_B, independently of its path, is summed up into the marked cell.

matrix can vary from total bytes and packets to the number of losses between two endpoints. Thereby, it can handle different representations of this metric, such as the total, average, maximum, minimum, etc. Traffic matrices usually consider all ingress and egress switches as nodes in a way that it records end-to-end traffic within a network. Traffic matrices have diverse applications such as pattern detection, traffic forecasting, anomaly detection, capacity planning, network provisioning, and more [Tun+13].

3.2.1 DistTM: *Centrally Coordinated Flow Monitoring*

This section is based on the distributed traffic matrix estimation system, denoted DistTM [Har+16]. It can be regarded as the distributed version of OpenTM [TGG10]. The following briefly describes the isolated (without coordination) traffic matrix generation within a network part controlled by a single controller.

Isolated Generation of Traffic Matrices

In order to generate the traffic matrix, DistTM requires knowledge about flows in the network. For this, we make use of SDNs main property, which is the concentration of all knowledge in the centralized controller. This allows the monitoring application to receive flow information from the routing application. The shared knowledge includes events about new flows, where flows traverse, and when flow rules expire[28]. When the controller installs a new flow, DistTM starts measuring it at a switch on its path. Tootoonchian et al. [TGG10] discuss different strategies to select a switch on the flow's path accordingly. Given knowledge about the path of a flow, the system can fill the traffic matrix cells by looking up the ingress as well as egress switch and updating the corresponding cell with the newest measurement. This easy-to-operate approach produces significant overhead in terms of conducted measurements. To overcome this, the measurement coordination, as described in the following, allows avoiding measuring each flow in each network by coordinating the controllers and sharing measurements on the control-plane.

Coordination to Eliminate Redundant Measurements

With regard to collaboration, we assume that the controllers are part of a distributed control-plane and, therefore, have a communication channel, such as proposed in Section 2.2.2 or 2.2.3. Figure 19 describes the architecture, roles, and communication pattern of DistTM and its coordination approach.

Depicted in the infrastructure layer, a flow F_1 traverses through all three subnetworks. Using a naïve approach, each controller measures it to update its traffic matrix. However, DistTM uses a coordinator, marked with a green-

28 OpenFlow-based networks with reactive routing generate so-called `PacketIn` messages when no flow rule matches a received packet. Thus, usually, they indicate a new flow. By analogy, switches dispatch so-called `FlowRemoved` messages if a flow rule expires [Pfa+12].

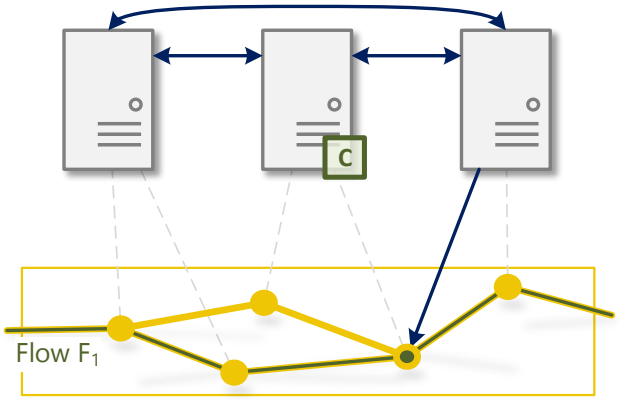


Figure 19: The DISTTM system uses a coordinator, marked with a green-rimmed "C" to assign measurement responsibilities to only a single controller for flows traversing multiple networks (here one flow denoted F_1). Measurements are subsequently shared on the control-plane with all interested controllers.

rimmed "C" in the figure, to assign the responsibility to measure F_1 only at a single controller. That controller also gets the responsibility to share the measurements with all interested controllers. In this case, all three controllers see the flow; hence, all controllers are interested in the measurements. One of the participating controllers takes the role of the coordinator so that it is responsible for the aggregation of monitoring tasks (cf. coordination entity in Section 3.1). To assign the coordinator role, we refer to leader election algorithms solving such problems [Awe87; Lam01; OO14].

Given all controllers agree on a coordinator and there is a usable communication channel between controllers, the coordination procedure, sketched in Figure 20, is as follows:

1. Whenever the controllers install a new flow in the network, DISTTM first starts measuring the flow independently of other controllers, thus possibly redundant in every controller.

2. The system increases an internal *flow measurement counter*.

3. When the counter within one controller reaches a predefined threshold value R, this controller requests a coordination from the coordinator[29].

4. Subsequently, the coordinator broadcasts interest requests to all controllers, including itself. The controllers answer by sending all new flow measurement interests that occurred since the last coordination. At that time, all controllers also reset the flow measurement counter.

29 Note that the controller itself can be the coordinator.

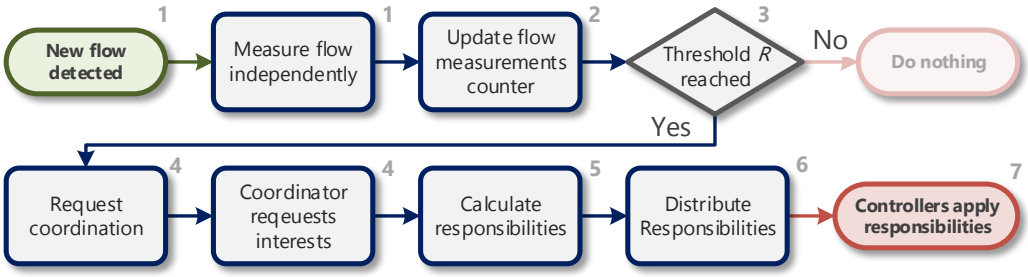


Figure 20: Sequence diagram for DISTTM's coordination workflow. Numbers on the upper right corner of each box refer to the steps in the enumeration. After a certain number of new flows (*threshold*), the coordinator requests measurement interests from all controllers and distributes responsibilities based on its knowledge. After each controller applies the responsibilities, each flow is only measured once and measurements are shared.

5. The coordinator uses some fairness scheme to calculate the measurement responsibilities per controller. Each flow measurement is assigned only once.

6. Consequently, the coordinator distributes the calculated responsibilities to the controllers, including information about other controllers interested in the particular measurements.

7. From that time on, the controllers only measure the flows they were assigned to and use received statistic updates from other controllers to update their traffic matrix.

As coordinations come with considerable overhead, we devise a *flow measurement counter* that allows controlling the frequency of coordinations. Setting the threshold to one triggers a coordination on every new flow. In contrast, setting it to a higher value, meaning that a coordination execution is done for multiple flows at the same time, shrinks the coordination overhead. Furthermore, it implies the advantage that very short flows (*mice flows*) that were removed in the meantime are not considered during the coordination.

As we use the coordination to fill traffic matrices, this work focuses on flow bandwidths (byte counter) as metric. The coordination works well with other path-agnostic metrics, which do not change on their path. Additive metrics, such as the latency or loss, cannot be coordinated as they are potentially different for each network, although they might relate to a shared resource (flow).

Fairness Efforts

While calculating the measurement responsibilities, the coordinator uses a fairness scheme to the balance monitoring load fair among the networking en-

tities. We propose three exemplary, combinable functions with paradigmatic objectives plus a random selection as reference:

Fair Controller Distribution (FCD): The fair controller distribution scheme targets to balance the load fairly among all controllers. To this end, the coordinator keeps track of the number of previously assigned and still active measurements per controller. To assign a new flow measurement responsibility, it looks up the controller with the least active measurements. With this, it considers only controllers that are interested in the flow, and thus have the flow traversing through their network in a way that it can measure it. Say m_c is the number of currently active measurements of a controller c from the set of controllers C. Then the responsibility for the next measurement is assigned to $c_{next} = \arg\min_{c \in C} m_c$.

Fair Domain Distribution (FDD): The fair domain distribution scheme shares load fairly among the controllers; however, it only considers the number of measurements of potential candidates for the flow measurements into account. As a consequence, if two controllers see a flow, it does not matter how many flows each controller shares with other network parts. Solely the ratio of measurements between involved parties matters. For example, consider controllers `A` and `B` share a large number of flows, while `B` and `C` share only few flows. A new flow that traverses only the networks of `B` and `C` will be assigned with equal probability to `B` and to `C`, although `B` has already a much higher load due to the shared measurements with controller `A`. Consider the set of controllers, which see a flow f_i, is $C_{f_i} \subseteq C$. Furthermore, the number of active measurements m_{c,f_i} of a controller $c \in C_{f_i}$ includes only those measurements that are shared with the whole set C_{f_i}. The measurement responsibility for a new flow f_i is then assigned to $c_{next} = \arg\min_{c \in C_{f_i}} m_{c,f_i}$.

Fair Switch Distribution (FSD): The fair switch distribution scheme takes account of the load distribution among switches of each network. To do so, the coordinator keeps track of the ratio between the total number of assigned measurements per network and the number of switches within each network. In consequence, a large network consisting of an order of magnitude more switches than other networks will be responsible to proportionally measure the equivalent higher number of flows targeting equal load per switch. Say $S_c = \{s_1, s_2, \ldots, s_N\}$ is the set of switches within the network of controller c. m_{s_i} is then the number of flows s_i currently measures. The controller responsible for the next measurement is then $c_{next} = \arg\min_{c \in C} \sum_{s_i \in S_c} m_{s_i}$.

Random Distribution (RD): As a reference for the other fairness schemes, the random distribution assigns measurements randomly to one of the participating controllers without considering any load. Ultimately, this will

distribute the load similarly to the distribution of flows in the networks. If flows are distributed equally throughout all networks, the random distribution resembles *FCD*. The controller that measures the next flow is $c_{next} = \mathsf{U}[1, |C|]$ with U being a uniformly distributed random variable.

Example

For a better understanding of DistTM, this paragraph contains a descriptive example. Consider a clean slate start using DistTM with a measurement threshold R (cf. Figure 20) of three. In accordance to Figure 21 and Table 2, first, flow f_A (purple) arrives and traverses only the subnetwork controlled by C_A. Next, f_B (orange) arrives, traversing subnetworks of C_B and C_C. When f_C (green), passing through C_A's and C_B's networks, arrives next, C_A and C_B have a flow measurement counter of two and C_C of one (cf. Table 2 in the third row). Now consider f_D (blue) arrives and passes through C_A's, C_B's, and C_C's network. Both controller C_A and C_B count three new flows, therefore reaching their measurement threshold.

At that time, both controllers request a coordination from the coordinator, who was elected beforehand. The coordinator subsequently asks C_A, C_B, and C_C for their measurement interests, which are shown on the left-hand side of Figure 22: C_A is interested in f_A, f_C, f_D; C_B in f_B, f_C, f_D; and C_C in f_B, f_D. Using this information the coordinator makes use of the fairness scheme of choice. In the following, assume it uses the fair controller distribution (*FCD*) scheme.

As Figure 22 reveals, the distribution is as follows: C_A measures f_A, as it is the only controller being able to capture f_A, as well as f_D. C_B measures f_C and C_C measures f_B. The distribution assigns an equal number of measure-

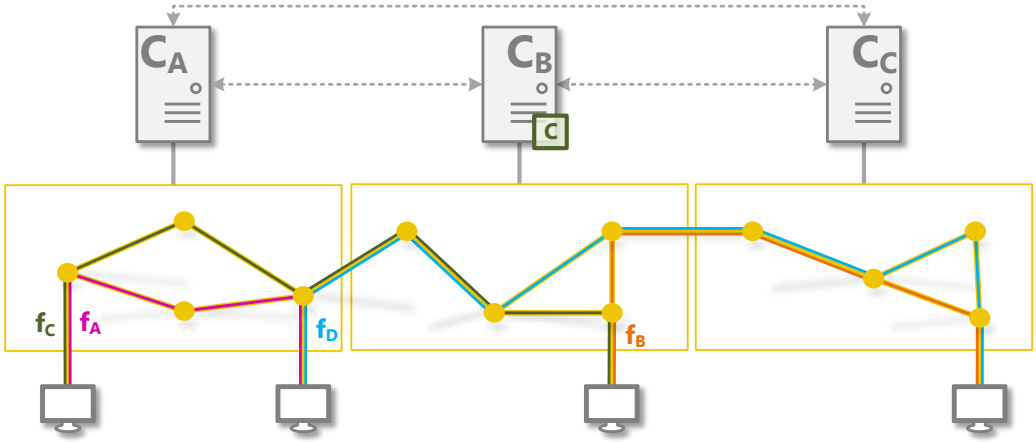


Figure 21: Exemplary DistTM scenario: Different flows traverse multiple adjacent networks such that their measurements can be done cooperatively, thereby avoiding redundant measurements of the same information.

Table 2: Overview of the flow arrivals and interested controllers for the example given in Figure 21. The righthand side shows the measurement counters at different times. Underlined counters have reached the threshold R and trigger a coordination request.

Time	*Arrival*	*Interested controllers*	*Counter* C_A	C_B	C_C
1	f_A	C_A	1	0	0
2	f_B	C_B, C_C	1	1	1
3	f_C	C_A, C_B	2	2	1
4	f_D	C_A, C_B, C_C	$\underline{3}$	$\underline{3}$	2

ment responsibilities to each controller as far as possible. f_D, which traverses all networks could also be assigned to C_B and C_C, depending on the implementation. After the responsibility calculation, the coordinator dispatches responsibility assignment messages to each controller telling them which other controllers are interested in their measurements (e.g., C_A must measure f_A and f_D while C_B and C_C are interested in measurements of f_D). The right-hand side of Figure 22 shows the information each controller receives.

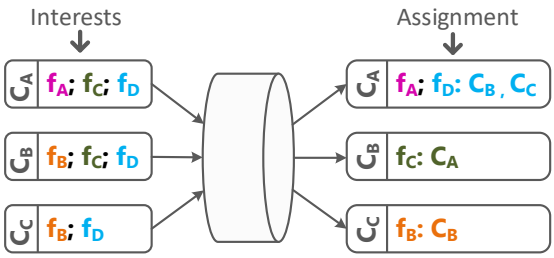


Figure 22: Input and output of the fair controller distribution (FCD) scheme according to the given example. The left-hand side shows the input of each controller, including the flows in which they are interested. The right-hand side shows the output, thus the responsibilities each controller got assigned (e.g., f_A; f_D). Additionally, it lists interested controllers for each measurement responsibility (e.g., f_D:C_B,C_C).

3.2.2 *Decentrally Coordinated Flow Monitoring*

The previously described approach coordinates flow monitoring among a federated SDN using a centralized coordinator. The role of the coordinator imposes a single point of failure, making it vulnerable against network partitioning or controller failures. To reduce this risk, we extend the design using decentralized coordination. This section is based on our work [Har+19b].

Although various existing consensus protocols, such as PAXOS [Lam01] and RAFT [OO14], consider fail-safety, their functionality exceeds the requirements here. Thus, to avoid unnecessary complexity and accompanied overhead, we develop a lightweight, distributed coordination algorithm to assign measurement responsibilities to controllers, which is triggered in each controller for arriving flows.

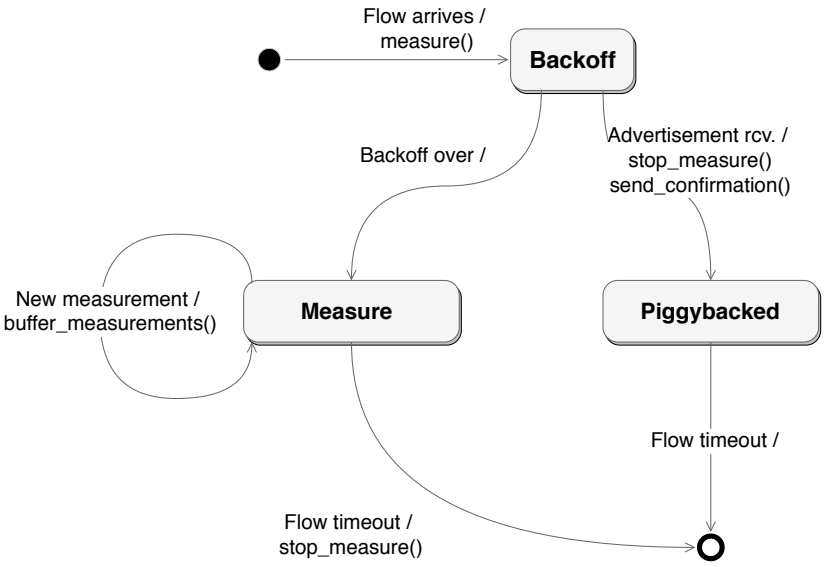


Figure 23: State diagram of the coordination algorithm: Whenever new flows arrive, the algorithm starts by going in the *Backoff* state. The first controller to reach its randomized backoff time traverses to the *Measurement* state and broadcasts an advertisement for this flow. Other controllers that receive the advertisement go to the *Piggybacked* state where they receive measurements from the controller in the *Measurement* state. The flow removal terminates the algorithm on all controllers.

Decentral Assignment Algorithm

Figure 23 summarizes an algorithm that consists of three states: *Backoff*, *Measurement*, and *Piggybacked*. As soon as a flow arrives, the algorithm changes into the *Backoff* state. In the *Backoff* state, the controller starts measuring the flow. At that time, usually, all controllers seeing the flow are in the *Backoff* state and the flow is measured in each of these controllers redundantly, as known from the centralized coordination approach (cf. Section 3.1). In this state, each controller calculates a randomized backoff time as it enters this state. We can adjust the backoff time calculation to enforce fairness among controllers, which we detail later. The controllers wait for the backoff time they generated and continue measuring the flow independently.

The controller's algorithm that reaches its backoff time first traverses to the *Measurement* state. When the algorithm reaches that state, it solicits for the responsibility to measure this flow on behalf of all controllers. To

inform other controllers about the flow measurement responsibility, the controller that reaches the *Measurement* state sends an advertisement message to all other controllers containing information on the flow. The controllers that are still in the *Backoff* state and receiving this advertisement go over to the *Piggybacked* state. As long as a controller is in this state, it receives measurements for the corresponding flow from the controller it received the advertisement from. Therefore, the controller stops measuring the flow on its own when reaching the *Piggybacked* state, eliminating the redundancy in the measurements. Furthermore, it confirms its interest in the measurements. Controllers that do not see the flow ignore the advertisement such that they receive no measurements for this flow and do not participate in the responsibility negotiation.

The controller in the *Measurement* state continues measuring the flow periodically and, in addition to processing it, e.g., updating its own traffic matrix, it sends the measured statistics to interested controllers. To do so, after conducting each measurement once, the controllers assembles a statistic message for each of the other controllers containing all measurements they are interested in. These messages summarize all measurements within one period, meaning that multiple measurements are convolved within a single batch message (cf. Figure 24). As a consequence, the total costs for shared measurements are significantly reduced in comparison to a redundant measurement of each flow with a dedicated statistics request and response.

The algorithm terminates when the flow expires. Considering OpenFlow networks, *FlowRemoved* messages inform each controller about this event. As long as the flow is active, periodic statistic updates serve as implicit keep-alive notifications. If a flow is still valid and statistic updates from the responsible controller stay absent, either the connection is lost, e.g., due to network partitioning or the responsible controller failed. In the last case, all controllers detecting such a failure go back to the *Backoff* state where they measure the flow by themselves and generate a new backoff time. If multiple controllers lost connection to the measuring controllers, a new coordination is automatically performed as all of them enter the *Backoff* state roughly at the same time.

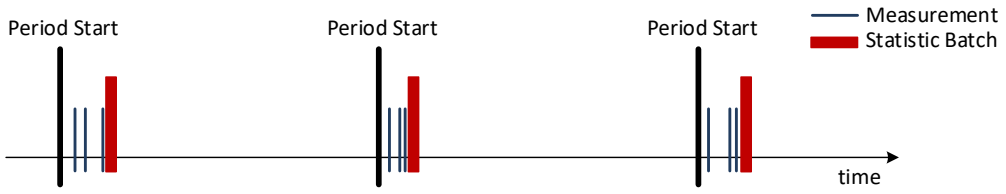


Figure 24: In each measurement period, the controller waits until all measurements are successfully conducted to send statistic batch messages to other controllers with statistics they have registered for.

Collision Avoidance

In the case that multiple controllers traverse into the *Measurement* state simultaneously, e.g., by calculating a similar backoff time, each of the controllers dispatches an advertisement for the negotiated flow. After sending, each controller receives the advertisement(s) of the other controller(s). Controllers that are still in the *Backoff* state and receive an advertisement will not go to the *Measurement* state, and thus will not send an advertisement. Nevertheless, controllers facing an advertisement after sending one detect a so-called *collision*.

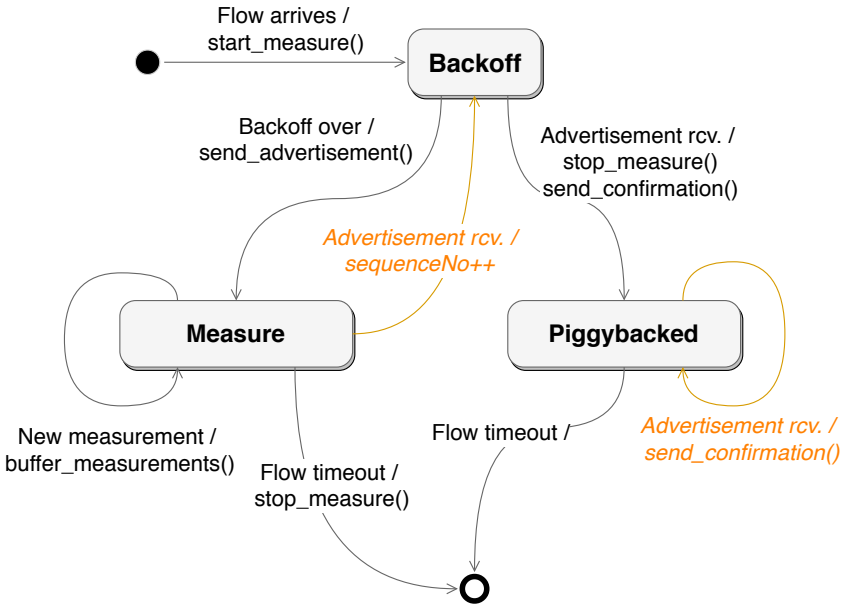


Figure 25: State diagram of the coordination algorithm with collision resolving: When the algorithm is in the *Measurement* state and receives another advertisement for the same flow, it detects a collision and traverses back to the *Backoff* state. A new coordination is conducted with all controllers detecting the collision. Other controllers always confirm the first advertisement with the highest sequence number published.

To avoid this, we use a `CSMA/CA` [Col83]-inspired calculation of the randomized backoff time. Before we describe how collisions should be avoided, we present an algorithm extension to resolve collisions. The state diagram in Figure 25 shows the extended algorithm. Consider a scenario where multiple controllers generate a close-by backoff time and traverse to the *Measurement* state. All of them send an advertisement for the negotiated flow to other controllers that are then in the *Piggybacked* state. Once the controllers in the *Measurement* state receive another advertisement and, therefore, detect the collision, they traverse back to the *Backoff* state (cf. Figure 25). In that state, they calculate a new backoff time. The other controllers remain in the *Piggybacked* state already receiving statistics from the controller they confirmed their interest to. The controller with the lowest new backoff time goes into

the *Measurement* state again and sends new advertisements for this flow. However, we introduce a sequence number to advertisements, which is incremented on each collision, to indicate that all previous advertisements are obsolete. Controllers that are still in the *Backoff* state confirm their interest and controllers, which are already in the *Piggybacked* state, reconfirm their interest due to the higher sequence number. This procedure is performed recursively until only a single controller remains in the *Measurement* state and the algorithms have eliminated the measurement redundancy.

Fairness Efforts

We can adjust the calculated backoff time targeting fairness. The algorithm considers the ratio r_i of measurement responsibilities m_i of the controller i compared to the total number of responsibilities among all controllers m_{total}, with N being the number of controllers:

$$r_i = \frac{m_i}{m_{total}} = \frac{m_i}{\sum_{j=1}^{N} m_j}. \tag{2}$$

If a controller i only has a low portion of measurements ($m_i << m_j \; \forall \; j \neq i$), it has a high chance of being responsible for the next measurements. Vice versa, if a controller i already measures a significant portion ($r_i >> \sum_{j=1, j\neq i}^{N} m_j/N$), it will have a low chance of being responsible for the negotiated measurement. In order to have the required information about the portion of measurements, all controllers include the number of measurements they are responsible for in each advertisement, confirmation message, and statistic update during the coordination. This information is used to continuously update the total number of active measurements of all controllers (m_{total}). As for the backoff time, the algorithm first calculates a reference backoff time b_{ref}. It is calculated through Equation 3.

$$b_{ref} = F \cdot m_i^2 \cdot m_{total}^{-1} \tag{3}$$

The reference backoff time is calculated by multiplying a configurable factor $F \in \mathbb{R}_+^*$ with the ratio m_i^2/m_{total}. The configurable factor F has a twofold meaning. First, it is the trade-off between the responsiveness of the algorithm and collision probability. A small F allows the algorithm to go to the *Measurement* state more quickly; yet, it will render collisions more likely. Second, short *mice* flows will be ignored if the factor exceeds the flow's lifetime. Coordinating measurements for *mice* flows is not reasonable due to the disproportionality between coordination overhead and saved costs from redundancy elimination. For the evaluation, we set the factor by default to three seconds. Nevertheless, the selection of a proper value depends on the network conditions and the operator's requirements. We take the square of the number of

assigned measurements m_i to enhance large ratios and depress small ratios. Accordingly, b_{ref} increases exponentially such that the reference backoff is disproportionally large if a controller already has more load. The final backoff time b orientates by the reference backoff time b_{ref}:

$$b \sim \mathcal{N}(b_{ref}; 0,1 \cdot b_{ref}). \tag{4}$$

Given in Equation 4, b is a random sample taken from a normal distribution with the mean $\mu = b_{ref}$ and $\sigma^2 = 0,1 \cdot b_{ref}$ as the variance. Intuitively, this means that the lower the references backoff time, the lower the actual backoff time. Consequently, the lower the backoff time, the higher the probability of being the first controller whose backoff time ends or, in other words, the higher the probability of being responsible to measure the negotiated flow. We multiply b_{ref} by 0,1 with the notion to reduce the risk that controllers with a low portion of measurements will randomly select a high backoff time due to the random distribution. Nonetheless, in case there is no single controller with a much smaller portion than others, the variance is accordingly larger. Thus they will not pick close-by random backoff time with a high chance. Figure 26 shows this behavior.

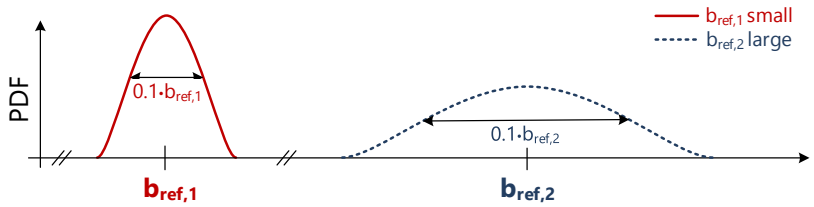


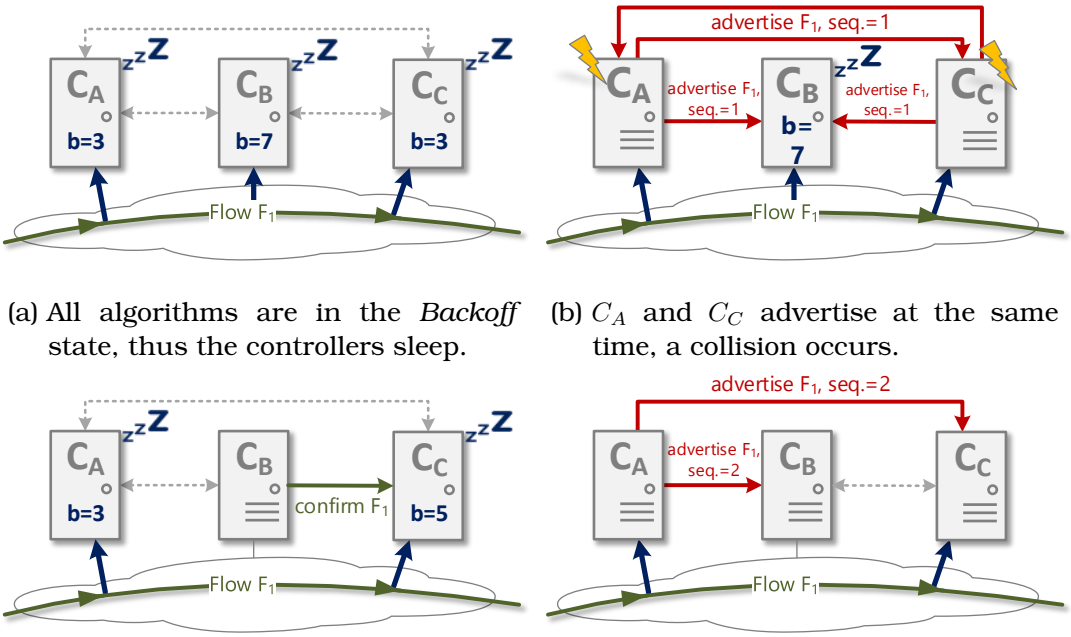
Figure 26: The probability density function for the backoff time b, given that b_{ref} is the mean and also influences the variance. Large reference backoff times allow a wide range of values for the backoff time around b_{ref} as mean, but small reference backoff times limit the spreading around b_{ref}.

Consider a controller that is responsible to measure a small number of flows compared to two other competing controllers. The controller calculates a reference backoff time, which is much lower and, as the variance is accordingly small, the randomly chosen backoff time will be close to the reference backoff time. Subsequently, the probability is very high that he is responsible to measure the next negotiated flow. The other way around, if one controller has significantly more flows to measure, e.g., because it sees more flows on average than the others, it will generate a comparably larger reference backoff time and pick a large backoff time, respectively. The probability of having the lowest backoff time, which would lead to traversing in the *Measurement* state and being responsible for the measurements is vanishingly low. The described adjustment of the randomized backoff time calculation allows en-

forcing fairness among controllers in situations where some controllers see more flows than others.

Example

In the following, we provide an example that describes the workflow of the decentrally organized coordination extension of DISTTM. Consider a newly arriving flow F_1. Each controller detecting the flow calls the coordination algorithm, which immediately goes into the *Backoff* state where it generates a randomized backoff time depending on the ratio of measurements per controller. In the example, the controllers sleep for three (C_A), seven (C_B), and again three (C_C) seconds as given in Figure 27a.



(a) All algorithms are in the *Backoff* state, thus the controllers sleep.

(b) C_A and C_C advertise at the same time, a collision occurs.

(c) C_B confirms the first advertisement. C_A and C_C resolve the collision by going back to the *Backoff* state.

(d) Only C_A advertises F_1 as it reaches its backoff time first. The collision is resolved.

Figure 27: Exemplary workflow for the decentralized coordination. In Subfigure a, the controller's algorithms are in the *Backoff* state. In Subfigure b, the controllers with lowest backoff times advertise F_1, thus leading to a collision. C_B confirms the first advertisement (here of C_C); however, C_A and C_C go back to the *Backoff* state due to the collision. In Subfigure d, only C_A advertises F_1, which is confirmed by both C_B and C_C afterwards.

As a consequence of similar backoff times, C_A and C_C reach the end of their backoff time simultaneously. Thus, following Figure 27b, both controllers advertise to measure this flow to C_B and each other. C_B confirms the first advertisement it receives, to C_C in this case. C_C subsequently starts sending periodic updates for F_1 to C_B and C_B stops its measurements of F_1. As C_A

and C_C detect the collision by receiving an advertisement after the algorithm traversed to the *Measurement* state, thus after sending an advertisement, both go back to the *Backoff* state, as shown in Figure 27c. Hence, both wait for another backoff time, now three seconds for C_A and five for C_B. In the next subfigure, 27d, only C_A advertises to both other controllers, after reaching its backoff time first. As the new advertisement contains a higher sequence number than the previous, C_B will reconfirm its interest to C_A. Also, C_C now confirms its interest to C_A such that only one controller is responsible to monitor F_1. C_A shares its measurements periodically using batch messages that also include other statistical updates of other flows according to the interests of C_B and C_C.

3.3 Evaluation

This section contains the evaluation of the three approaches to coordinate the monitoring in controllers of federated SDNs. First, in Section 3.3.1, we describe the environment, scenarios, and methodology used in this evaluation. This includes the investigated data-plane topologies and respective control-plane organization. Furthermore, for each scenario, we describe the applied traffic, particularly the flow characteristics. Section 3.3.2 introduces a set of four hypotheses, which describe what should be achieved with the proposed approaches. Subsequently, in compliance with the order of the hypotheses, Section 3.3.3 highlights the main result, which is the cost reduction due to eliminated redundancy in measurements. For the different approaches, we show *(i)* that we generally reduce the number of required measurements and *(ii)* the behavior against different system parameters such as the update period and coordination threshold. Section 3.3.4 shows the trade-off when considering the additionally added overhead for coordination. Furthermore, Section 3.3.5 shows the effect of the fairness efforts with respect to controller and switch load. Lastly, the accuracy of the cooperative monitoring is revealed in Section 3.3.6 while considering the staleness of measurements.

3.3.1 *Evaluation Environment, Methodology, and Configurations*

We conduct a simulative evaluation to show quantitative results. To be able to evaluate using realistically large topologies with certain sizes, a simulative environment is chosen. Furthermore, the majorly examined metrics, such as packet and byte counts, do not suffer inaccuracy in simulations. For the evaluation, we use MININET [LHM10] to deploy a virtual SDN network on a powerful physical machine[30]. MININET uses software switches, so-called OPEN VSWITCHES [Pfa+15b], which support OpenFlow. The controllers run as separate Java processes on the server that also hosts MININET. In addition,

30 Ubuntu 16.04 x64 server; 24 Intel(R) Xeon(R) cores @ 2,60GHz; 128GB Memory

we use the Floodlight[31] OpenFlow controller. Floodlight allows extending the controller with additional modules and provides APIs to applications. We implement the cooperative traffic matrix estimation application as controller modules and the dedicated coordination entity proposed in Section 3.1 as Floodlight controller.

Result Illustration Methods

Results are given as Cumulative Distribution Function (CDF), Complementary Cumulative Distribution Function (CCDF), Box-Whisker-Charts, or bar charts. CDF's show the distribution of an investigated metric. The depicted function in the CDF charts is $P[X \leq x]$, meaning that the value at x is the probability that the investigated metric takes a value less or equal to x. In the case of a measured data set, it is the portion of values lower than x. The CCDF is complementary to CDF and, therefore, $P[X > x]$. In combination with a logarithmic y-axis, it can, for instance, be used to highlight the tail of a distribution. The Box-Whisker-Charts (or Box-Plots) reports the median of a distribution, the first and third quartiles (25% and 75% of the distribution). Whiskers report the first datum greater than the first quartile minus $1,5 \cdot IQR$ (IQR is the interquartile range) and the last datum less than the third quartile plus $1,5 \cdot IQR$. Furthermore, outliers are marked for values outside of this range. In rare cases, the charts neglect "impossible" outliers that are assumed to originate from implementational issues. Bar charts show the mean of a data set as well as the 95% confidence intervals if not stated differently. All charts include at least 30 independently obtained data points for each metric displayed. Measurements of the DistTM system include at least 50 data points.

Investigated Topologies

We consider two topologies for the following evaluation. First, a synthetic topology allows showing the principles of the measurement redundancy elimination. This topology consists of nine linearly connected switches, each connected to one host that is capable of sending and receiving traffic. Three controllers share this topology, whereas each controller controls three directly connected switches. On top of this, we use a distributed data center scenario as motivated in Section 2.2. As depicted in Figure 28, multiple interconnected data center networks that are controlled by a dedicated physical controller each, create a federation. Despite monitoring, the controllers provide shortest-path routing and other basic configuration functionality. In the example given in the figure, the structure of a single data center follows [Roy+15]. Multiple *Fat-Cat* switches are interconnected as a ring. The next layer is organized in clusters. Therefore, each cluster consists of multiple *Cluster* switches that are again connected in a ring. In the representative topologies, the *Fat-Cat* ring

31 Project Floodlight: OpenFlow Controller http://www.projectfloodlight.org/floodlight, accessed 16 April 2019

as well as each cluster have two switches such that the ring becomes a single link. Each *Cluster* switch connects to all *Fat-Cat* switches. The next layer consists of *Rack* switches. These are not interconnected; however,they are again connected to each *Cluster* switch within their cluster. In addition, each *Rack* switch connects to all servers within the rack. In the exemplary topology shown in Figure 28, a cluster has two *Rack* switches, therefore, two racks. Each rack has a single host representing the rack servers. The figure shows two such data centers that are connected through their first *Fat-Cat* switch. If mentioned in the evaluation, we connect an additional data center to the first *Fat-Cat* switch of the second subnetwork so that three SDN-controlled data centers set up the federation.

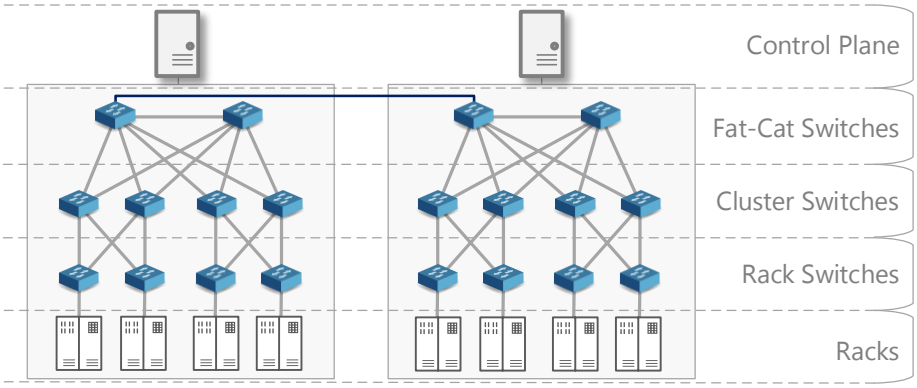


Figure 28: Interconnected data center networks using the layered structure of Facebook's data center [Roy+15]. The first layer consists of *Fat-Cat* switches connected to form a ring. The second layer contains *Cluster* switches for each cluster interconnected in a ring and connected to each *Fat-Cat* switch. The third layer has *Rack* switches that aggregate all rack server connections. Each *Rack* switch connects to all *Cluster* switches within its cluster.

Model-based Traffic Generation

For the evaluation, we generate flows between two randomly chosen hosts from the racks or, in case of the linear topology, any host connected to the linearly connected switches. We model the traffic aligned with data center traffic characteristics based on [Roy+15; BAM10]. In all evaluations, we use an exponentially distributed flow inter-arrival time such that the flows arrive as a Poisson process [Ros+96]. The flow length is selected from either Zipf's or a Pareto distribution, which are both power law with negative exponents scaled to have a cumulative distribution of one. Note that a Zipf distribution can be derived from Pareto distribution. Equation 5 gives the probability den-

sity function of the Pareto distribution with parameters $k > 0$ as scale and $x_{min} \geq 0$ as start value.

$$f(x) = \begin{cases} \frac{k \cdot x_{min}^k}{x^{k+1}} & \text{for } x \geq x_{min} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Both distributions lead to the generation of a dominant portion of short *mice* flows and rare long *elephant* flows, depending on the parameters used. For some evaluations, the flow length is used as an independent parameter such that the distribution is different. As the actual traffic load does not influence the evaluated metrics but only the monitoring application, we majorly use light traffic with either constant bit rate traffic or following a positively truncated normal distribution with a mean of 0,5 MBps and a standard deviation of 0,1 MBps. The parameters vary depending on the configuration:

The goal of the first configuration is to show how measurement aggregation works in a synthetic setup using the described simple linear topology. It is used for the evaluation of the DISTTM system. The exponentially distributed flow inter-arrival time uses a parameter $\lambda = 1$; thus a new flow arrives on average every second. Furthermore, the flow duration, or, in this case, the flow length in packets, follows a Pareto distribution with $k = 5$ and $x_{min} = 500$. In combination with a constant packet rate of 100pps and minimal, constant packet sizes, flows live for at least five seconds plus a five second flow idle timeout. In the following, we refer to this configuration as *Configuration A*. It's purpose is to show a proof of concept for the coordination mechanisms before we evaluate it using a more realistic configuration.

The second configuration, denoted *Configuration B*, uses the realistic federated data center topology. In this configuration, the flow durations in seconds follow Zipf's distribution with $s = 1,6$ as characterizing parameter. We truncate the durations to a maximum of 10^3s in order to guarantee that all flows terminate in a reasonable time. The flow inter-arrival time is exponentially distributed with $\lambda = 10$. The model and parameters are derived from [Roy+15; BAM10] and predominantly give flows with a lifetime of around a second with a few exceptionally long flows, which is common in data centers. Flows arrive as a Poisson process with a mean arrival rate of 0,1s and a median of $\lambda^{-1} \ln(2) \approx 0,07s$. This configuration aims to make traffic as realistic as possible. It is used to evaluate the first approach with a centralized coordination entity described in Section 3.1 and the extended DISTTM approach. For the latter, the mean flow duration is the independent parameter and follows, for the sake of determination and simplicity, a normal distribution with a standard deviation of five seconds. Instead of two interconnected data centers, here, we use three federated data center networks.

3.3.2 *Hypotheses*

This section describes four Hypotheses, which express the contribution's goals. To start with the superior goal, the first hypothesis targets the elimination of redundant measurements and the accompanied cost reduction:

H_1: *The coordination mechanisms reduce redundant measurements with the result that the costs for measurements can be decreased significantly.*

The second hypothesis considers overhead introduced by the coordination mechanisms and claims that the efficiency increases nonetheless:

H_2: *The overhead, which is generated for coordination is small, with the result that the overall monitoring costs decrease.*

The third hypothesis concerns the fairness efforts:

H_3: *The coordination mechanisms improve the fairness with respect to the load on involved entities such as controllers and switches.*

The last hypothesis targets the performance of coordinated monitoring:

H_4: *The performance does not suffer using coordinated monitoring mechanisms.*

In the following, we organize the results according to the formulated hypothesis. Thus, the next section shows the cost reduction in terms of required measurements. The subsequent section introduces coordination overhead costs and its trade-off with respect to the measurement cost reduction. Section 3.3.5 includes the fairness evaluation results and the last section, 3.3.6, results on the performance, namely timeliness of measurements. In each section, we denote the used environment and configuration.

3.3.3 *Redundancy Reduction*

The main target of our monitoring coordination is to aggregate measurement tasks from different controllers. Aggregation of measurement tasks leads to the elimination of redundant measurements. Therefore, we expect a reduction of the total number of measurements. In the following, we first show the reduction of the number of required statistic requests to measure the link between two data centers. For this, we apply *Configuration B*. The centralized correlator aggregates the monitoring tasks of two controllers.

Figure 29 shows the distribution of the total number of required statistic requests to measure the link bandwidth every second. With redundant measurements, the total number (shown in solid blue) concentrates mainly around 220 measurements. In contrast, the solid red curve reveals that task aggregation to eliminate redundant measurements saves about 50 percent of all measurements (mostly ~110 statistic requests required). This result shows exemplarily how the aggregation approach reduces costs.

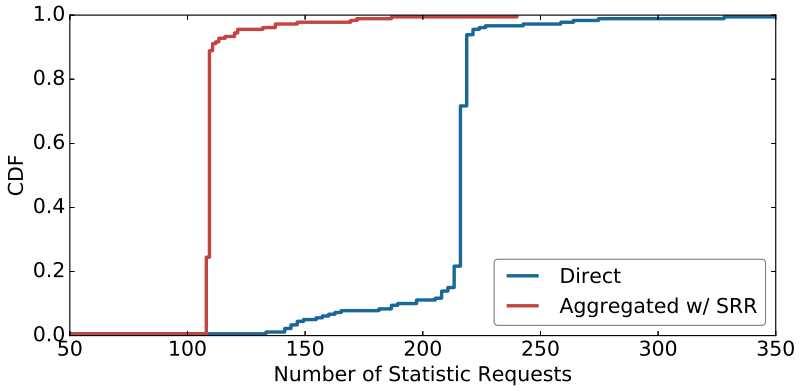


Figure 29: Distribution of required statistic requests to measure a shared link with (*Aggregated w/ SRR*) and without coordination (*Direct*). The coordination aggregates overlapping monitoring tasks such that it saves roughly half of the measurement costs [Har+18a].

The next evaluation shows a similar metric; however,not using a central entity but DISTTM. Furthermore, Figure 30 shows how the coordination threshold, more precisely, the parameter that allows configuring the frequency of coordinations, controls the cost reduction. The results shown in Figure 30 are collected using *Configuration A* and the measurement period is two seconds.

First, the light-blue dashed curve with crosses, showing the distribution of the number of statistic requests without coordination (w/o), displays that the number of statistic requests required to measure all flows in the network dominates all runs with coordination. Redundant measurements lead to the highest number of required statistic requests. Using the coordination of DISTTM, e.g., with a threshold of one ($R = 1$), shows that the total number can be significantly decreased. When increasing the threshold, thus coordinating less often, we observe that the costs for measurements increase again towards the curve without coordination. We will show later that the threshold also controls the overhead for coordinations and, therefore, selecting the lowest threshold is not the best choice ultimately. For the sake of completeness, the results of the cost reduction when using the decentralized coordination algorithm can be found in Appendix A.2.

The following result shows how the measurement frequency influences the aggregation mechanism. As the centralized coordination entity has capabilities to aggregate monitoring tasks with partly different specifications, such as

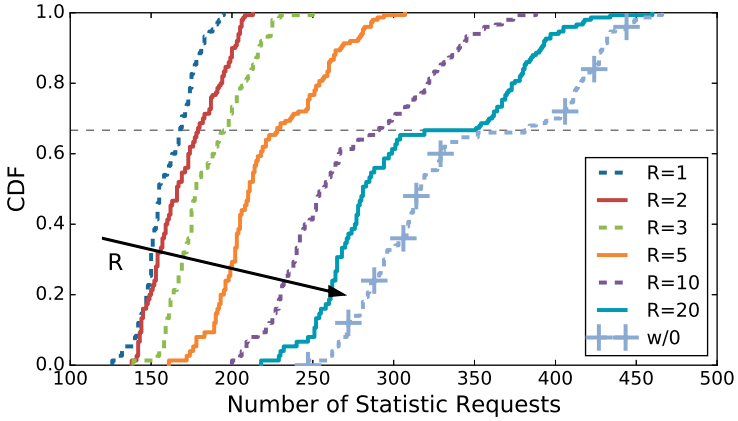


Figure 30: Distribution of required statistic requests to measure all flows in the network with different coordination thresholds ($R = 1 \dots 20$) and without coordination (w/o). The lower the threshold, the lower the costs as coordinations come more frequently. Without coordination and, therefore, with redundant measurements, the costs are the highest [Har+16].

the measurement frequency, it is used for this evaluation. Furthermore, *Configuration B* is used in this case. Both controllers measure all flows traversing their networks. Thereby, they allow the correlator to change the measurement point such that monitoring tasks of disjunct subnetworks can be aggregated. The first controller measures with a static frequency of $f_A = 1/s$, whereas the second controller's measurement frequency varies.

Figure 31 shows the number of required statistic requests for direct measurements without coordination (*Direct*, in green with hashes) and measurements using the central coordination entity that aggregates both measurement tasks (*Aggregated w/ SRR*, shown in blue without hashes). The x-axis lists varying measurement frequencies (f_B) of the second controller. If both controllers measure with the same frequency of $f_A = f_B = 1/s$, the costs reduce from ~10,5k to ~7,5k statistic requests in median. Decreasing the frequency f_B to $1/2s$ decreases the uncoordinated costs as the second controller measures only half as often. Yet, when aggregating, the cost reduction is lower since all flows that traverse the first subnetworks must be measured once per second anyway. This trend continues for higher frequencies. Accordingly, the aggregation mechanism saves relatively more costs if the frequencies are equal.

In Summary, the results presented so far show that the coordination approaches reduce the total measurement costs by aggregating redundant measurements. They support the first hypothesis H_1.

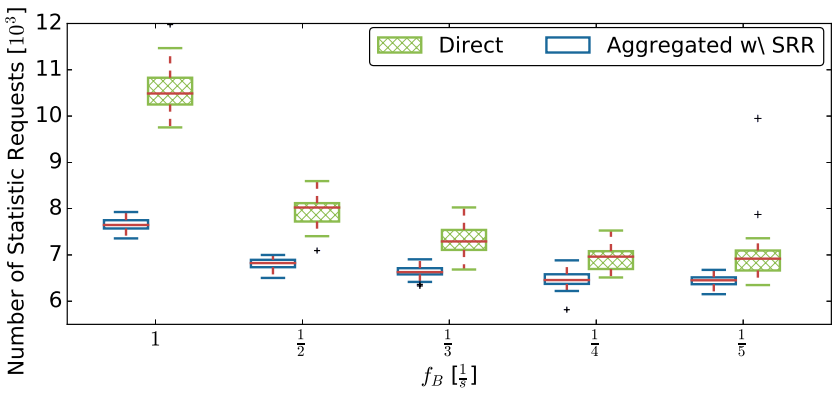


Figure 31: Required statistic requests to measure all flows in the network with changing measurement frequencies of one controller. If both controllers have the same frequency of $f_A = f_B = {}^1/_s$, the costs reduce the most through aggregation. The relative cost reduction reduces if the frequencies diverge [Har+18a].

3.3.4 *Coordination Overhead Trade-Off*

The second hypothesis H_2 claims that the additionally generated overhead for the coordination is limited and, therefore, the overall costs reduce. To investigate the sustainability of H_2, in this section, we quantify the coordination costs and their influence on the total costs. Figure 32 depicts three box plots: *(i)* the leftmost subplot contains the costs for statistic collection as already shown in the previous section for comparability; *(ii)* the innermost subplot shows the additional costs for communication during the coordination process of DISTTM; and *(iii)* the rightmost subplot contains the sum of both. All three subplots contain the costs in terms of bytes for statistic or coordination messages per controller per flow. The measurements were conducted using DISTTM, thus in *Configuration A*. The x-axis shows different coordination thresholds as an independent parameter. Note that the costs for DISTTM with a coordination threshold of one (x-axis) approximate the costs of the extended DISTTM system with decentralized coordination described in Section 3.2.2 under the assumption that no collisions occur.

Considering only the leftmost plot with the statistic collection costs, we recognize the results from the previous evaluation section again. Without coordination (*w/o*), the costs are the highest with a median of around 1100 bytes per flow. Using the coordination scheme, we observe that the costs per flow are the lowest with a median of around 500 bytes. Increasing the threshold increases also the costs again until a threshold of 20 flows, which almost reflect the costs without coordination. However, considering the coordination overhead in the innermost plot, the figure shows, of course, zero bytes for the naïve, uncoordinated approach. Having a threshold of one, hence, conduct-

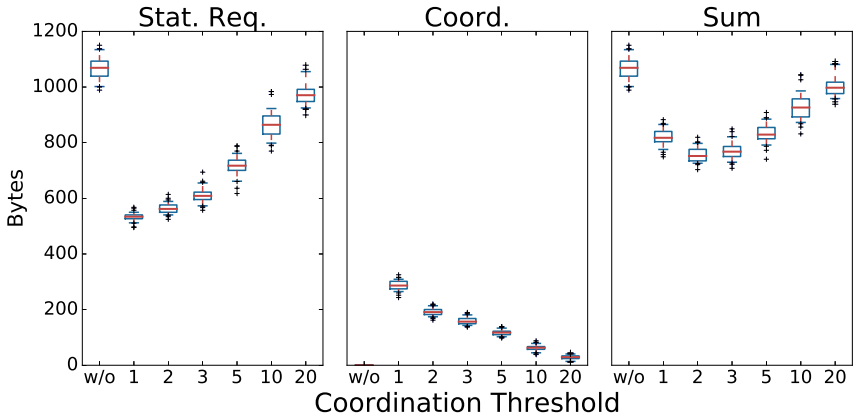


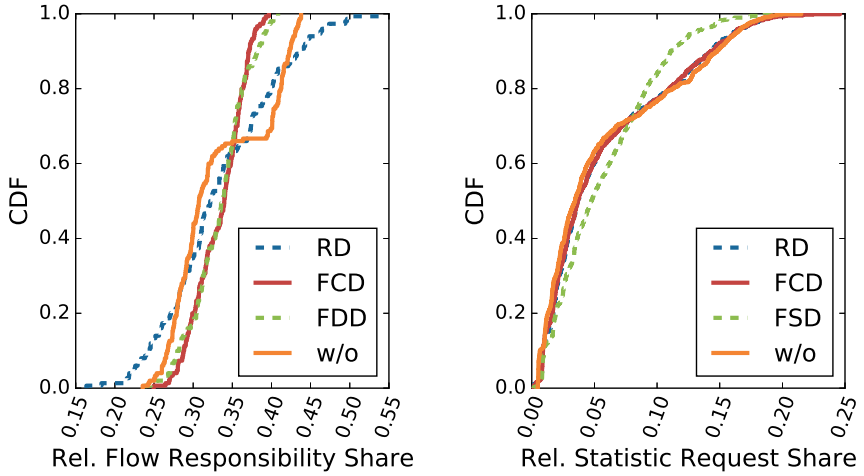
Figure 32: The leftmost plot shows the measurement costs in bytes with different coordination thresholds and without coordination. The innermost plot depicts the coordination overhead in bytes and the rightmost plot the sum of both. The figure shows how the threshold controls the trade-off between overhead and cost reduction [Har+16].

ing a coordination after each new flow or using the decentralized algorithms leads to the highest coordination overhead. Higher coordination thresholds scale the costs down as coordinations happen less frequently. Finally, if the threshold is 20, we have almost no overhead as coordinations become very rare. The rightmost plot shows this trade-off clearly. Taking the sum of the collection and coordination costs reveals that the threshold allows controlling the utility. Note that the sum of both costs is only an intuitive measure as the different cost types cannot be compared directly. Costs for statistic collection happen between the controllers on the control-plane and the switches on the data-plane, whereas the costs for coordination stay within the control-plane between controllers. Simply adding both costs is only an intuitive illustration. In the investigated setup, coordinating after each new flow is not optimal. Here, the most efficient configuration uses a threshold of two or three. Although we have to take the sum of collection and coordination costs with caution, the results show that the total costs decrease when also considering additionally generated coordination overhead. This holds true for DistTM and its extension with decentralized coordination. As a consequence, the results support hypothesis H_2. Appendix A.3 contains further results considering coordination overhead with different measurement intervals.

3.3.5 *Fairness Improvement*

The third hypothesis (H_3) aims towards fairness among controllers and switches, respectively. It claims that the coordination approaches help balancing load fair among the entities. The following results were measured using the DistTM system and *Configuration A* as well as the extended DistTM

system with decentral coordination in *Configuration B*. Figure 33 shows the distribution of the portion of measurements p_i each controller (or switch) i is responsible for: m_i/m_{total}. The steeper the curve, the more controllers share a similar load (portion of measurements).



(a) Distribution of measurement cost ratios among controllers.

(b) Distribution of measurement cost ratios among switches.

Figure 33: Fairness evaluation results. Depending on the selected fairness distribution scheme (cf. Section 3.2 for acronyms), DistTM can balance the load fair among controllers or switches. Subfigure a shows that the controller load fairness increases using *FCD* or *FDD* while Subfigure b shows the switch load fairness improvement using *FSD* [Har+16].

Figure 33 shows the cumulative distribution of the relative flow responsibility share of each controller (Subfigure a) and switch (Subfigure b). The baseline, given in solid orange, shows the distribution without using DistTM (*w/o*). The figure shows that around 66% of all portions p_i are between 0,25 and 0,33. Afterwards, the remaining ~33% are between 0,39 and 0,44. This behavior comes from the fact that the outer two of the three controllers in the linear topology see on average fewer flows than the controller in the middle. Recalling *Configuration A*, the flow's origins and destinations are picked randomly with uniform probability. Thus, the controller responsible for the inner three switches sees on average more flows than the other two controllers. Therefore, also the relative share of measurement responsibilities is unfairly distributed without coordination according to the centrality of the controllers. In the ideal case, considering three controllers, all portion values are exactly $r_i = 1/3$ so that each of the three controllers is responsible for exactly one-third of the measurements. Looking at the random distribution (*RD*) scheme, given in dashed blue, the step after $2/3$ shrinks. Here, DistTM's coordinator randomly assigns monitoring responsibilities to one of the controllers seeing the flow. As a consequence, the random distribution curve is not as fair as the

load-balancing schemes; however, it does not assign more measurements to the controller in the focal position. Using a more sophisticated scheme, such as the fair controller distribution (*FCD*) or fair domain distribution (*FDD*), the fairness improves further. The distribution using the *FCD*, depicted in solid red, is comparably steep between 0,3 and 0,36 for most values with only small tails. The distribution using *FDD*, dashed green, follows a similar pattern. Thus, it can be concluded that all controllers have a similar share of the measurements in the system when using the DISTTM with fairness schemes. Discrimination of central controllers vanishes completely.

Furthermore, considering the relative share per switch, Figure 33b reveals the distribution of the measurement share per switch among all networks. For this evaluation, we enlarged the synthetic topology of *Configuration A* so that the third subnetwork consists of four times the number of switches than the other (12 instead of 3). Without coordination (solid orange, *w/o*), DISTTM with the random distribution scheme (dashed blue, *RD*), and DISTTM with a controller oriented fairness scheme (solid red, *FCD*) behave similarly, as shown in the figure. In all three distributions, about 30% of the switches have a higher load than the other switches as, observable with a changing gradient of the curve at 0,6 (y-axis). Taking into account that the schemes distribute measurement tasks based on the controllers, each network measures roughly the same number of flows. Subsequently, in networks with fewer switches, the load per switch, as shown in the figure, is comparably higher. In contrast, the fair switch distribution scheme (*FSD*), shown in dashed green, includes the number of switches in each network into the task assignment. Thus, using this scheme improves the fairness among switches. We observe that the dashed green curve is steeper and does not have any steps. A perfect distribution is not possible as switches in focal positions see on average more flows and only switches seeing a flow are candidates for measurements.

Equation 6, denoted Jain's Fairness Index [Jai90], is a fairness score between $1/n$ (worse) and 1 (best). We use the index to quantify the fairness just described in Table 3.

$$f(x_1, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} = \frac{\overline{\mathbf{x}}^2}{\overline{\mathbf{x}^2}} \tag{6}$$

The fairness index reports equal controller fairness performance using *FCD* and *FDD*, which dominate a random distribution and the non-coordinated case, as already observed in Figure 33a. With respect to the switch fairness and using the adapted topology with an increased number of switches in the last subnetwork, Table 3 indicates that *FSD* achieves the highest fairness among switches. *FSD* is the only fairness scheme that considers the switch number and outperforms all other schemes with respect to switch load.

32 Adapted topology: the last subnetwork of *Configuration A* contains 12 instead of three switches.

Table 3: Jains Fairness Index for controller and switch load [Har+16].

Scheme	*Controller Fairness*	*Switch Fairness*[32]
RD	0,9510	0,5377
FCD	**0,9893**	0,5400
FDD	**0,9907**	—
FSD	—	**0,6736**
w/o DISTTM	0,9677	0,5413

Furthermore, we evaluate the fairness using extended DISTTM with decentralized coordination. The decentral coordination algorithm targets to enforce fairness by adjusting the random backoff times based on the number of measurement responsibilities compared to all active measurements in the network (cf. Figure 23, p. 47). Figure 34 depicts the distribution of the ratio of measurements $p_i = {}^{m_i}/_{m_{total}} = {}^{m_i}/_{\sum_j m_j}$ per controller (m_i is the number of measurements of controller i) with *Configuration B* and three data centers, hence, also three controllers.

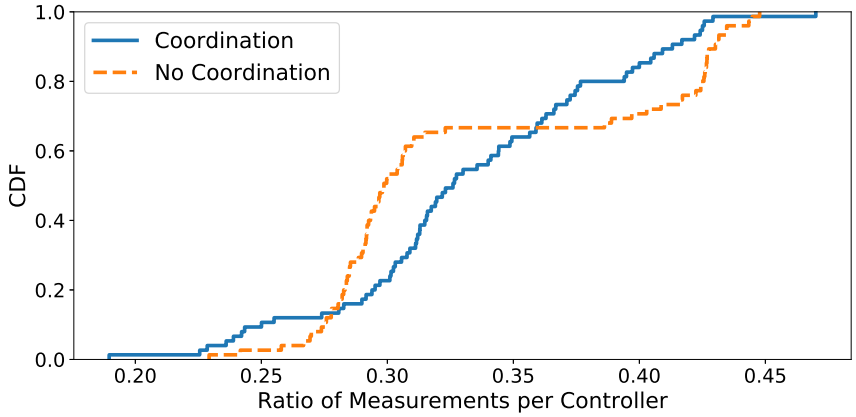


Figure 34: Distribution of the ratio of measurement among controllers. Without coordination, the controller in the middle is responsible for more measurements than the others and the dashed orange line has a step after ~⅔. Using the cooperative approach, the step vanished so that the controller in the center is not discriminated anymore [Har+19b].

The dashed orange curve shows the distribution without coordination and redundant measurements. It can be observed that after ²/₃ a step in the distribution occurs. We assume that this is due to the focal position of the controller responsible for the data center in the center as it sees more flows than the outer controllers when flows are uniformly distributed among all racks. Using the cooperative approach that distributes responsibilities based on the ratio of measurements, the solid blue distribution curve shows a more even slope. This approach removes the step such that the algorithm does not dis-

criminate single controllers in central positions. Considering also that the overall distribution is more even but spread from roundabout 0,23 to 0,43, we conclude that the ratio of measurements a controller is responsible for (p_i) is picked from 0,23 to 0,43 with uniform probability.

The results found in this evaluation support hypothesis H_3, saying that the coordination improves fairness among controllers and switches. First, depending on the selected fairness distribution scheme, DistTM allows balancing the load fair among controllers and switches, respectively. A deductive fairness scheme combines both controllers and switches. Furthermore, the results of the decentralized coordination algorithm balances load fairer among the controllers using a randomized `CSMA/CA` [Col83]-inspired backoff time such that there is no discrimination of single controllers.

3.3.6 *Measurement Accuracy*

Lastly, this section investigates the performance using collaborative monitoring with a view to hypothesis (H_4). We argue that the actual byte counter values within a traffic matrix will not suffer in terms of accuracy as they are only measured from another controller. We assume the detection of potential losses, e.g., due to failures or overload, with separate monitoring mechanisms. Still, the time it takes for a value to be updated, which we denote *traffic matrix entry staleness*, can increase due to transmission delays and varying measurement times when sharing measurements.

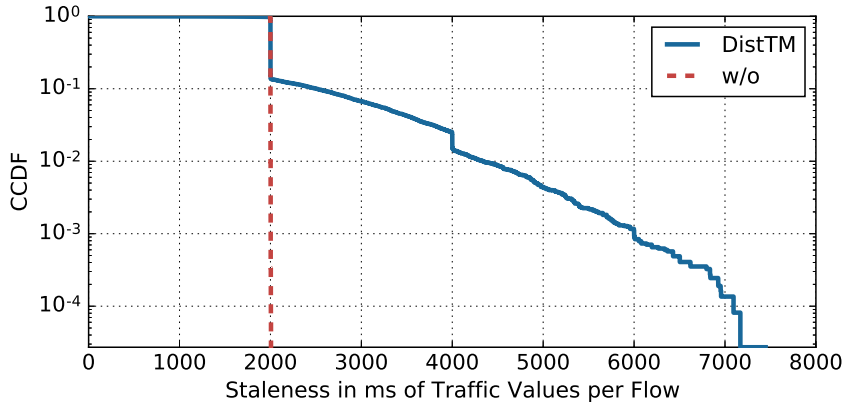


Figure 35: Staleness of traffic matrix entries. Almost 90% of all values come without timeliness penalty when using DistTM. About 99% of all values get updated in less than twice the time [Har+16].

Figure 35 shows the staleness of measurements in milliseconds. As a matrix entry can contain multiple flows, the figure gives the staleness for flow counter measurements. The results are collected using DistTM in combination with *Configuration A*. The configuration sets the measurement period,

thus the time between to consecutive measurements of the same counter, to two seconds. Note that the y-axis is logarithmic and the figure is a CCDF.

As the dashed red line depicts, the staleness of entries not using shared measurements (without coordination, *w/o*) is always equal to the update period. There is no penalty in the staleness when measurements are not shared on the control-plane and every controller measures itself. Considering shared measurements for the traffic matrix given by DISTTM, shown in solid blue, the figure shows that about 90% of all values come without any timeliness penalty (call in mind that the measurement period is two seconds). Afterwards, the staleness increases such that about 99% of the measurements come with less or equal twice the time since the last update (4000ms), hence, up to two seconds timeliness penalty. This 9% between 2000ms and 4000ms are first measurements after a successful coordination: Consider a controller conducts a measurement; yet, after almost two seconds a coordination assigns the measurements for this flow to another controller. Depending on the time the other controller started with its measurements, it can take at up to two further seconds until it conducts the next measurement and shares it. Thus, it takes up to four seconds until the measurement is updated. Subsequently, the timeliness should not suffer further for this flow. For the same reason, shorter update times are also possible as visible in the figure. The remainder has exponentially distributed excess staleness.

Having almost no timeliness penalty for 90% of all values and at most twice the measurement period for 99% supports the last hypothesis H_4.

3.3.7 *Evaluation Result Summary*

This chapter showed the evaluation results for the developed cooperative monitoring approaches in federated SDNs. First, within a synthetic scenario with a linear topology, we showed that the measurement task aggregation mechanisms reduce the total costs significantly while maintaining reasonable coordination overhead. The results were confirmed within interconnected data center topologies. As the found results support all four hypothesis, we conclude that the developed approaches *(i)* decrease the total cost for monitoring when network controllers collaborate; *(ii)* the coordination overhead is reasonable and the monitoring efficiency can be significantly improved; *(iii)* the fairness among network entities such as controllers and switches can be improved with sophisticated assignment of monitoring responsibilities; and *(iv)* that the monitoring performance in terms of data freshness does not degrade notably.

3.4 Summary

This chapter presented the first contribution of this thesis. It tackled the problem of monitoring in federated Software-Defined Networks. Federated

SDNs either are large-scale networks with a distributed control-plane, where every controller is responsible for a dedicated network part or are interconnected subnetworks, e.g., a distributed data center network. In such networks, controllers often monitor shared resources such that we proposed mechanisms to eliminate redundancy of measurements when multiple controllers (or applications) capture the same information. We presented three variants of a monitoring task aggregation solution: *(i)* A centralized entity that takes monitoring tasks from different controllers to unburden them from periodic measurements. The entity aggregates different tasks from different origins that overlap in their specification so that redundant tasks are eliminated. *(ii)* A distributed task aggregation system, denoted DISTTM, which leverages one controller as central coordinator. The coordinator identifies overlapping monitoring interests and instructs controllers to share their measurements with interested controllers such that no measurement is induced redundantly. *(iii)* Lastly, a distributed task coordination approach that uses a distributed algorithm to assign measurements to a single controller while others share their interests with the controller in charge in order to receive the measurements. The decentral algorithm is robust against controller failures and network separation. In the evaluation, we showed how the approaches significantly decrease the costs for measuring the same information when redundancy is eliminated. In comparison with the introduced overhead, we described the increase in efficiency and how system parameters allow optimizing the trade-off between coordination overhead and redundancy elimination. Finally, we presented that the timeliness of measurements remains almost unaffected.

4

Efficient Collection of Monitoring Data

So far, this thesis proposed approaches for collaborative monitoring in federated Software-Defined Networks. As to that, it focused on the coordination between controllers to monitoring task redundancies. This raises the question of how controllers perform the individual monitoring data collection. To this end, in this second contribution chapter, we tackle three elementary aspects of the collection process, which are not yet fully explored in related work and, therefore, open space to increase the efficiency of monitoring in SDNs. Figure 36 summarizes these aspects, which are detailed in the following.

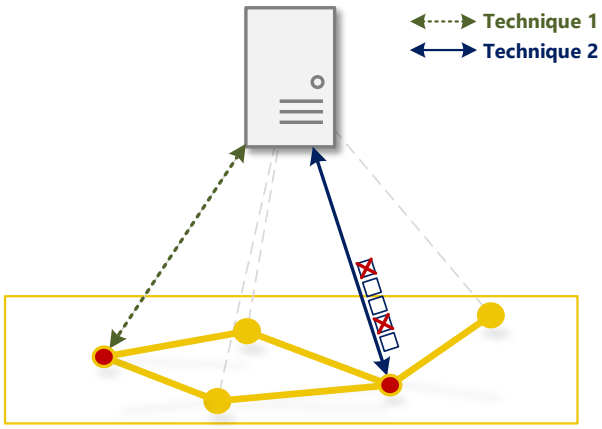


Figure 36: Three investigated aspects with respect to the data collection: *Where* to measure statistical information; *How* to capture the information; *Which* information to capture.

First, the control-plane, here abstracted as a single logical controller, can use different techniques to measure statistical information in the network. In addition to traditional measurement techniques, SDN provides a plethora of effective methods to measure the network state. In the figure, the controller uses *Technique 1*, depicted with a dashed green arrow, to measure statistics at the leftmost switch. Furthermore, it uses *Technique 2*, shown as a solid blue arrow, to measure information on another switch. The first aspect tackled in this chapter is:

1. *How to measure statistical information?*

Second, as the figure shows, the controller measures information at two switches that are marked red. The location where information is captured plays a major role when investigating the efficiency of monitoring. While some measurement points provide highly valuable information, others only add little gain to the measurement quality such that the trade-off between costs and use is imbalanced. Therefore, the second aspect investigated in the chapter, as existing works not yet cover all requirements, is:

2. *Where to measure statistical information?*

Lastly, the third investigated aspect, sketched next to the solid blue measurement arrow, is the selection of measurement information. Despite where and how to measure, the question of what to measure arises. Regarding that aspect, in this thesis, we target to select the measured information based on its use for the application. Since not all measurements improve the quality of the applications, the efficiency might suffer. The corresponding, third aspect of this chapter is:

3. *Which statistical information to measure?*

Each of the following sections covers one of the aspects of monitoring information data collection. Within each section, we clarify why the current state-of-the-art does not fulfill all requirements and motivate the need for further contributions. Section 4.1 tackles the first aspect, namely, how to measure information. In this section, instead of developing yet another technique to fit all situations, we show that different techniques have different advantages based on network conditions and measurement requirements. We use that knowledge to increase the efficiency of monitoring using an adaptive approach. In Section 4.2, we treat the second question of measurement location selection. We propose a solution to intelligently select measurement points in dynamic networks to boost the information value from small numbers of measurements. Lastly, Section 4.3 tackles the third aspect of the selection of statistical information. We develop an approach to select the measurement information based on its use for the issuing applications. Doing so, we save costs for communication and processing to further increase the monitoring efficiency. As we design the approaches in each section to be applied independently of each other, the evaluation results do not influence each other. Therefore, each section contains the evaluation of the respective contribution.

4.1 Adaptive Measurement Technique Selection

A plethora of works proposes monitoring approaches to capture statistical information using new SDN technologies [Yu+15; AB16; vDK14; Yu+13]. In ad-

dition, techniques from legacy networks can potentially be reused or adapted within SDNs [PCD03; AMM98; Sav99]. However, due to the dynamic nature of today's networks and ever-changing demands, these techniques face varying conditions over time. Although many efforts has been put into the optimization of individual techniques, for example, smart probing pattern for sampling-based monitoring [Bac+07; Bac+07; Boz+16; Mac+07], none of the techniques from this huge pool dominates in every situation. By analogy, existing monitoring frameworks usually use only one technique [vDK14]. We show in the following how exemplary techniques behave under different conditions and requirements. Based on the gained knowledge, we propose to switch between techniques instead of optimizing a single technique or just adapting parameters. The concept of transitions to switch between mechanisms in communication networks has been proven useful [Gro+13; Ric+15b; Ric+16b]. This section is based on [Har+17b].

For the sake of presentation, we decide to use an exemplary monitoring task, namely the measurement of packet loss and, as a consequence, failures. Almost any type of network temporarily exhibits packet loss due to failing components, misconfiguration, or overload [PCD03; Pax97; YKT99; ZD01]. Excessive packet loss can be depicted as one of the worst service quality disruptions as communication channels temporarily become completely unusable. To show the performance of different techniques, we select four representative loss measurement methods ranging from classical techniques to techniques designed for Software-Defined Networks, as well as possible techniques in future networks.

4.1.1 *Representative Loss Detection Techniques*

This section introduces four representative loss detection techniques. In this thesis, the term *technique* denotes a sequence of operations involving different networking entities in combination with different data formats, which produce a certain information, i.e., here packet loss ratio of a link. The term *packet loss ratio* is interchangeable with *packet loss probability*. However, assuming the ratio of losses (intensity of the stationary *loss/no loss* process $\{l_n\}_{n\in\mathbb{Z}}$) to be equal to the loss probability of a single packet ($\mathrm{E}[l_0] = P_L$) is only true if the loss process is stationary and ergodic [BB13].

Legacy Packet Counters (LC)

The first technique of choice uses plain packet counter. Packet counters are a prominent example for a technology SDN brings into focus; however, in legacy networks monitoring protocols, such as SNMP [WHP99], provide packet counters as well. As the solid blue line in Figure 37 depicts, the loss on a link can be measured by fetching the packet counters for the corresponding out-port at the ingress switch and the in-respective port at the egress switch. Let $c_s(t_i)$ be the cumulative packet count at switch s at time t_i. $s \in \{in, eg\}$ is either the

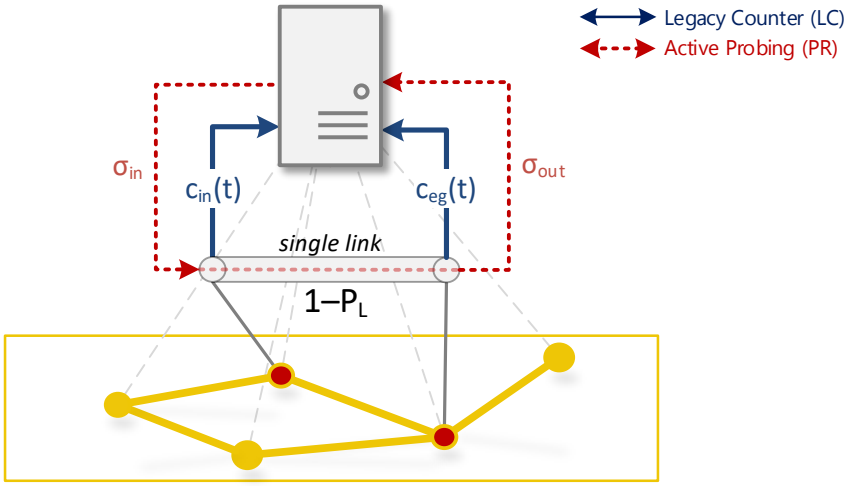


Figure 37: Schematic use of the *Legacy Counter* (*LC*) and *Active Probing* (*PR*) techniques to estimate the loss ratio. For *LC*, the controller fetches packet counters from the ingress and egress switch of the questioned link to calculate the number of lost packets. For *PR*, the controller sends probes through the investigated link and counts the number of lost probes.

ingress or the egress switch. For a certain time interval $[t_{i-1}, t_i]$ the number of lost packets L_i, measured using the byte counter, is

$$L_i = \underbrace{[c_{in}(t_i) - c_{in}(t_{i-1})]}_{\substack{\text{packets in } [t_{i-1},t_i] \\ \text{at ingress switch}}} - \underbrace{[c_{eg}(t_i) - c_{eg}(t_{i-1})]}_{\substack{\text{packets out } [t_{i-1},t_i] \\ \text{at egress switch}}}. \tag{7}$$

To estimate the loss probability $\hat{P}_{L,i}$ during that interval, we calculate

$$\hat{P}_{L,i} = \frac{L_i}{c_{in}(t_i) - c_{in}(t_{i-1})} = 1 - \frac{c_{eg}(t_i) - c_{eg}(t_{i-1})}{c_{in}(t_i) - c_{in}(t_{i-1})}. \tag{8}$$

This technique is easy to apply and intuitive. However, it faces two limitations affecting its accuracy: First, the time at which packet counts are taken at the ingress and egress switch may vary due to jitter in the control path between the controller and switches. Also, the load on a switch may affect the response time of a switch [BR13]. Furthermore, the egress counter does not include packets ”in-flight,“ i.e., currently queued. Under friendly, constant traffic conditions these effects are negligible and packets in-flight are carried

to the next interval such that the bias equalizes. Additionally considering the aforementioned timing delays, we calculate the loss probability with

$$\hat{P}_{L,i} = 1 - \frac{c_{eg}(t_i + t^{\epsilon}_{i,eg}) - c_{eg}(t_{i-1} + t^{\epsilon}_{i-1,eg})}{c_{in}(t_i + t^{\epsilon}_{i,in}) - c_{in}(t_{i-1} + t^{\epsilon}_{i-1,in})}. \tag{9}$$

In-line Packet Counters (IC)

This technique is comparable with the previous legacy counter technique, thus sketched in Figure 37 as solid red arrows. However, instead of actively pulling packet counters, control messages that are sent either way between the switches and controllers piggyback packet counters (cf. [Yu+13]). This method aims to minimize the communication costs for statistic retrieval. The main difference to the legacy counter technique is the indeterminism of counter update times. Monitoring applications can neither influence the time nor the frequency of arbitrary control messages. Solely a minimal update time is guaranteed through OpenFlow's *echo* keep-alive messages. Times between 10 and 20 seconds are common[33]. Due to that, we do not compare packet counters directly, but instead compare throughput (packets per time) of the ingress and egress switch of the investigated link or segment. The link's throughput at the ingress switch (*in*) is estimated with

$$\hat{R}_{in}(t_{i-1}, t_i) = \frac{c_{in}(t_i) - c_{in}(t_{i-1})}{t_i - t_{i-1}}. \tag{10}$$

The time distances between two consecutive counters updates at t_i and t_{i-1} are non-equidistant. The loss probability can now be calculated comparing the estimated bandwidths with

$$\hat{P}_{L,i} = 1 - \frac{\hat{R}_{eg}(t_{k-1}, t_k)}{\hat{R}_{in}(t_{i-1}, t_i)}. \tag{11}$$

Equation 11 is calculated whenever the egress switch sends a counter update. Therefore, $t_k \geq t_i$. The time difference between both counters is the limiting factor of the accuracy. Considering it as t^{Δ}_k in the equation by setting $t_i = t_k + t^{\Delta}_k$, we find

$$\hat{P}_{L,i} = 1 - \frac{R_{eg}(t_{k-1}, t_k)}{R_{in}(t_{k-1} + t^{\Delta}_{k-1}, t_k + t^{\Delta}_k)}. \tag{12}$$

33 Dell OpenFlow Deployment and User Guide 4.0 *Dell Software-Defined Networking (SDN)*, "echo-request interval," 2017 [Online] `https://www.dell.com/support/manuals`, accessed 03 May 2019

Sampled Packet Counter (SC)

Numbers of works, such as [Mos+15; Mos+14; YJM13], propose more efficient counting techniques as designated in OpenFlow, majorly using sketches. However, these techniques trade accuracy for costs. We assume that such a technique is available and abstract it with imprecise so-called sampled counters. A sampled counter does not count all packets within a flow but only increases the counter with a certain probability. Despite the packet counting, the technique works similar to the legacy counter technique so that controllers actively request counter.

We take samples of incoming packets in the ingress and egress switch stochastically according to an independent and identically distributed (iid) Bernoulli random variable, which can take the values {0,1}. 0 is *not sampled* and 1 is *sampled*. At the ingress, we sample a packet with the probability p_{in} and at the egress with p_{eg}. Thus, the total number of incoming packets at the ingress switch in $[t_{i-1}, t_i]$ is estimated with

$$\hat{c}_{in}(t_{i-1}, t_i) = \frac{c^*_{in}(t_{i-1}, t_i)}{p_{in}}. \tag{13}$$

Here, $c^*_{in}(t_{i-1}, t_i)$ is the number of sampled packets. Using the estimated number of packets at the ingress switch and egress switch allows estimating the loss probability in analogy to the legacy counter approach with

$$\hat{P}_{L,i} = 1 - \frac{\hat{c}_{eg}(t_{i-1}, t_i)}{\hat{c}_{in}(t_{i-1}, t_i)} = 1 - \frac{c^*_{eg}(t_{i-1}, t_i) \cdot p_{in}}{c^*_{in}(t_{i-1}, t_i) \cdot p_{eg}}. \tag{14}$$

The dominant limitation of this technique is the accuracy of the counter values. The lower the sampling rate, the lower the counting costs but the lower also the accuracy. Setting both p_{in} and p_{eg} to 1 reconstruct precise counter.

Active Probing (PR)

This technique is fundamentally different from the previous; however, widely used [Yu+15; AB16; BS04]. The controller sends (and receives) probe packets to measure how the network affects the stream. As shown as dashed blue arrow in Figure 37, p. 72, the controller measures the loss ratio by sending a number of packets σ_{in} through the ingress switch on the investigated link. Comparing this number to the number of received packets σ_{eg} at the egress switch allows calculating the number of lost packets and, therefore, to estimate the loss probability. The estimation for interval $[t_{i-1}, t_i]$ is trivial using

$$P_{L,i} = 1 - \frac{\sigma_{out,i}}{\sigma_{in,i}} \tag{15}$$

at the end of the interval ($t = t_i$). Although the technique seems straight forward, the optimization space through different probe pattern is tremendous [Bac+07; Mac+07]. The predominant drawback of the active probing technique is its cost. The probes introduce load on the productively used channels. This is particularly true when probe bursts are used, which might artificially introduce loss and, therefore, biases the loss probability.

Note that each of the described techniques has plenty of space for optimization. However, our goal is not to tweak a single technique but to show how different techniques behave in different situations.

4.1.2 *Performance Comparison of Different Techniques*

The four described techniques constitute a representative set of techniques available in future SDNs, based on existing tools and easily realizable functions using programmable data-planes. This section shows how each of them behaves in different traffic conditions as well as with different measurement specifications. We implement the techniques using Open vSwitches [Pfa+15b] that support OpenFlow and sFlow [PL04], and a Floodlight[34] OpenFlow controller. As we evaluate the techniques with respect to their measurement accuracy, we desire the environment to be as realistic as possible. To this end, we deployed them in the GENI [Ber+14b] testbed and investigated a logically isolated Internet2[35] link between a rack at Stanford University and the University of Illinois. The *tc* Linux tool helped to control the link behavior, precisely, the loss profile. To implement the legacy counter (*LC*) technique, we use plain OpenFlow features, which are available out-of-the-box in Open vSwitches and Floodlight. For the in-line counters (*IC*), we model switch-to-controller control traffic as a Poisson process and trigger a counter requests with exponentially distributed time distances using $\lambda = 15$. We limit the time between consecutive inter-control messages to 15 seconds as this is a realistic OpenFlow echo request interval (cf. Section 4.1.1) in the absence of other control messages. Furthermore, the technique smooths the estimated throughputs to avoid jumps if occasionally frequent counter updates occur. To do so, the estimated throughput $R'(t)$ at time t is flattened using an *Exponentially Weighted Moving Average (EWMA)* so that the averaged bandwidth $R_{MA}(t)$ considering the newest measurement $R'(t)$ is given with

$$R_{MA}(t) = \alpha \cdot R'(t) + (1 - \alpha) \cdot R_{MA}(t - 1). \tag{16}$$

34 Project Floodlight: OpenFlow Controller http://www.projectfloodlight.org/floodlight, accessed 16 April 2019.

35 The Internet2 community: enabling the future `https://www.internet2.edu/about-us`, accessed 16 May 2019.

Here, α is the smoothing factor, which we empirically optimize to 0,2. It controls the responsiveness versus stability trade-off within the moving average. To implement the sampled counter (*SC*) technique, we utilize sFlow to sample packets with a probability of $p_{in} = p_{eg} = 2^{-4}$ based on results found in an auxiliary evaluation (see Appendix A.4). Counting all sampled packets constructs an inaccurate counter. The active probing (*PR*) technique uses OpenFlow's *PacketOut* to dispatch a packet from the controller through a data-plane link and *PacketIn* messages to receive the probe again.

In the following, we first show how the techniques perform in different traffic conditions, particularly traffic intensities. Subsequently, we show how the techniques react when the loss rate is non-stationary. Therefore, we emulate loss rate jumps in predefined time frames. Afterwards, we evaluate the technique's performance when having loss bursts in combination with different traffic patterns.

Traffic Intensity Sensitivity

To investigate the behavior of the techniques with different traffic intensities, we set the traffic rate to 10^2, 10^3, 10^4, and 10^5pps. Despite that, the traffic send over the link follows a Poisson process with exponentially distributed packet inter-arrival times. Figure 38 shows the difference between the measured loss probability $\hat{P}_L$ and the theoretical loss probability P_L set with the Linux traffic control tool *tc*, thus the error $|\hat{P}_L - P_L|$. Each subfigure contains the accuracy for one traffic intensity. Note the logarithmic y-axes. Under mild conditions with a traffic rate of 10^2pps, Figure 38a shows that *PR* and *IC* perform slightly better than *LC*. However, all three deliver results with a median error of around 1–2%. In contrast, *SC*, the sampled counter technique, produces poor results with a median error of over 10%. When increasing the traffic rate intensity (Subfigures b, c, and d), we predominantly see an improvement for *LC* and *SC*.

In addition, the sampling-based counter technique *SC* delivers usable results, e.g., for failure detection, under heavy traffic conditions. Even though *LC* further improves sharply with more traffic, meaning that counter-based techniques generally improve with higher packet numbers, we see that *PR* and *IC* do not improve similarly. Mainly, *PR* uses its own probes measures independently of production traffic so that it is always applicable. However, with a high traffic intensity, additional probes might bias the loss on the link, e.g., if the loss is introduced only because of a probe burst.

Loss Jump Sensitivity

In this paragraph, we assess the measurement technique's responsiveness. For this, after a fixed time, the loss probability increases strongly to reflect a failure in the network. As Figure 39 depicts, during one evaluation run, at $t = \{73, 124, 185, 226\}$ seconds, P_L jumps between 0% and 20%. The measure-

ment update frequency is 5s. The traffic rate is 10^3pps with exponentially distributed packet inter-arrival times. As observable in the figure, *SC* (dashed green) delivers poor accuracy under these conditions as already found in the latter investigation. Only a vague notion that the measured loss follows the theoretical P_L is possible. Looking at *LC* and *PR* (solid blue and dashed red), we find a good accuracy. Considering the responsiveness, after a loss rate jump, both techniques cannot react immediately, but after a second measurement interval, the accuracy is satisfying again. Nicely visible, particularly in $t = [73, 124]$, is the lousy responsiveness of the in-line counter technique

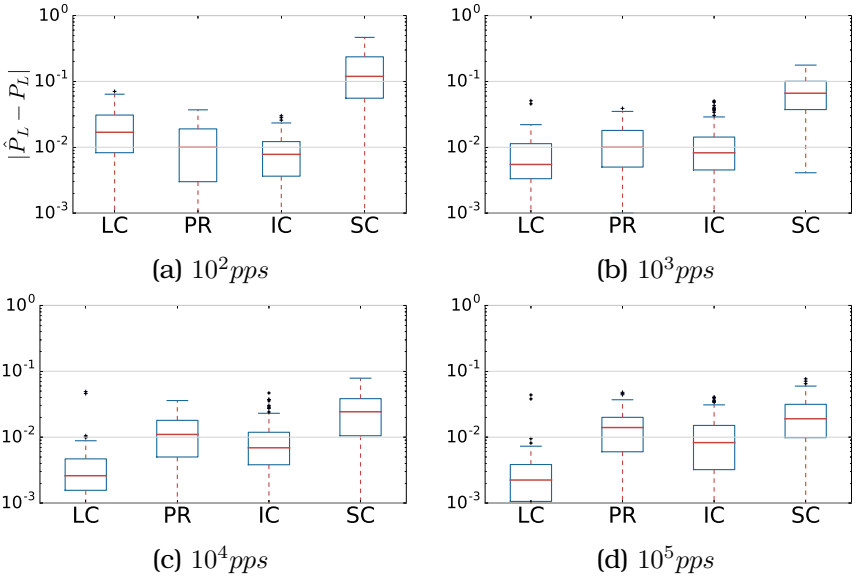


Figure 38: Error of the different techniques under different traffic intensities. With low traffic *LC*, *PR*, and *IC* show an error of around 2%. Increasing the traffic intensity improves *LC* as well as *SC*. *PR* and *IC* perform well under all traffic intensity conditions [Har+17b].

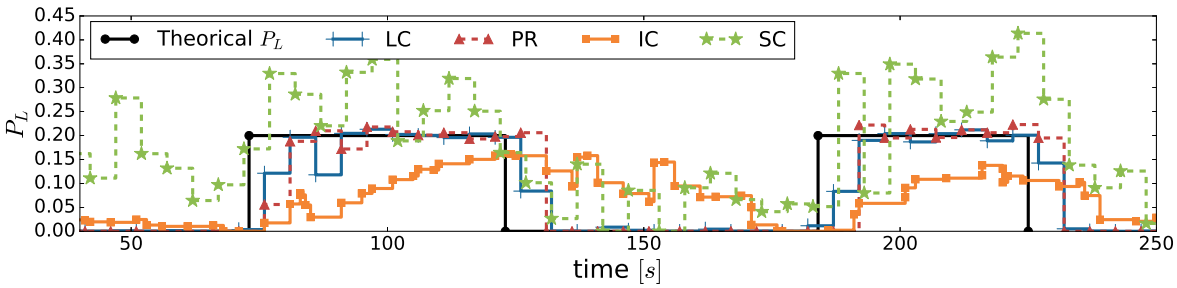


Figure 39: Timeline showing the loss probability estimations compared to the theoretical loss probability to assess the responsiveness of the techniques. *LC* and *PR* align fast with the theoretically true P_L, *IC* needs longer due to its bandwidth smoothing and *SC*'s measurements are in this configuration rarely close to the theoretically true P_L [Har+17b].

(*IC*). Due to the smoothing, which is required to retain stability in the situations of constant loss, the technique needs many counter updates until it reaches a reasonable accuracy. Recall that the operator has no influence on the counter update times in *IC*.

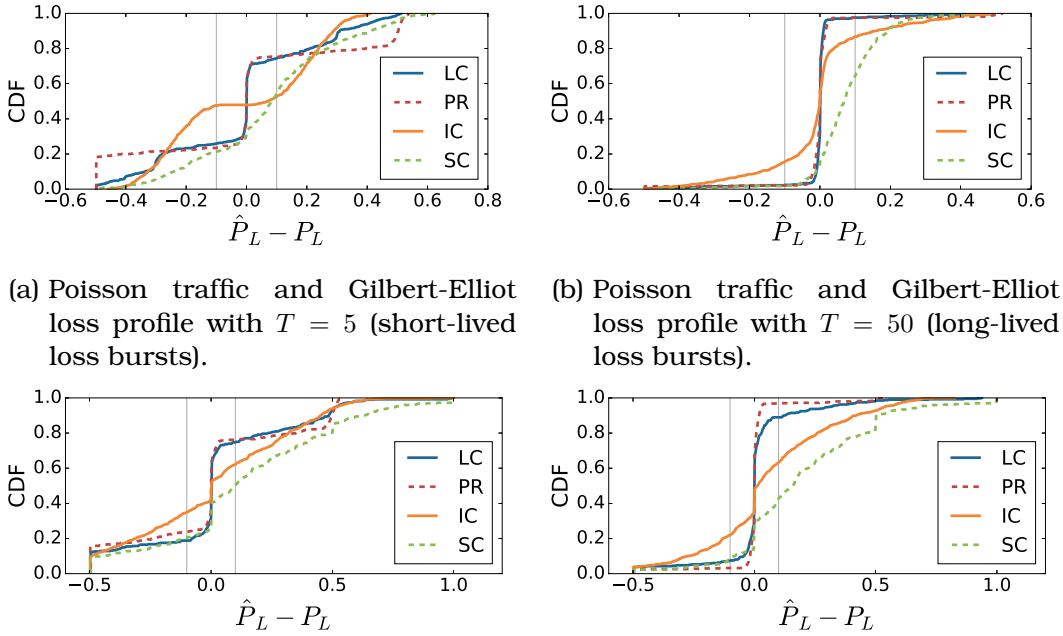
Traffic Type Sensitivity

This paragraph focuses on the technique's behavior for different traffic types in combination with different loss patterns. So far, we used friendly Poisson traffic such that we additionally investigate bursty traffic conditions. Furthermore, we investigate how the techniques accuracy changes with non-constant loss. The loss on the measured link is modeled using the Gilbert-Elliot 2-state discrete time model [HH08] with time slots of length three seconds. The loss alternates between zero and $P_{L,max} = 0,5$. The average loss rate is $P_{L,avg} = 0,25$ and the burstiness parameter T of the Markov chain varies within the experiment. T controls the burstiness, given a fixed $P_{L,max}$ and $P_{L,avg}$. Precisely, it defines the amount of time in which the chain changes its state twice (small T: short loss phases; large T: long loss phases).

Figure 40 first shows the error between $\hat{P}_L$, thus the measured loss probability, and P_L, the theoretical loss probability. The upper two plots depict "friendly" Poisson traffic, whereas the lower two plots have bursty traffic with uniformly distributed burst length of $\mathsf{U_L}[1, 10^3]$ packets and uniformly distributed inter-burst times of $\mathsf{U_L}[1, 10]$ seconds. While the left two subfigures have short-lived loss bursts $T = 5$, the two rightmost subfigures have long-lived loss bursts $T = 50$. Appendix A.6 contains results for $T = 25$.

Subfigure 40a shows that the accuracy with friendly traffic and short loss bursts is quite bad for all techniques. Particularly the in-line counter technique (*IC*) is not capable of reacting quick enough to trace the fast-changing loss probability and floats in the middle between zero and $P_{L,max}$. Also the other techniques need too much time to converge to the theoretical P_L. With less frequent loss changes, Figure 40b shows how the accuracy of almost all techniques increases. Particularly *LC* and *PR* show commendable accuracy, whereas *SC* has an offset, thus tends to overestimate the loss[36]. With bursty traffic and short-lived loss bursts, Figure 40c depicts poor accuracy for most techniques again. Using *LC* and *PR* produces in such very dynamic scenarios, at least in over 50% of all cases errors under 10%. Figure 40d shows the accuracy for bursty traffic and longer loss phases. It is visible that *LC* and *PR* improve significantly and can handle dynamic loss and traffic patterns. Comparing Subfigure b with Subfigure d lets conclude that *IC* is usable with moderate loss dynamics when the traffic is rather friendly. However, both subfigures reveal that *SC* is only usable with stable traffic and loss profiles. All techniques lack reliable accuracy when the loss profile is very dynamic. Taking measurements with a higher frequency can counteract this issue.

36 Further investigation of that effect showed that the sampling rate at the egress switch is biased under heavy load.



(a) Poisson traffic and Gilbert-Elliot loss profile with $T = 5$ (short-lived loss bursts).

(b) Poisson traffic and Gilbert-Elliot loss profile with $T = 50$ (long-lived loss bursts).

(c) Bursty traffic and Gilbert-Elliot loss profile with $T = 5$ (short-lived loss bursts).

(d) Bursty traffic and Gilbert-Elliot loss profile with $T = 50$ (long-lived loss bursts).

Figure 40: Cumulative distribution functions (CDFs) for the technique's measurement error with different traffic types (Poisson versus bursty) and loss phases with different burstiness [Har+17b].

Measurement Frequency Sensitivity

In this paragraph, we show how the techniques behave with different measurement update frequencies. Figure 41 shows the accuracy of the techniques while varying the update period. Appendix A.5 contains a corresponding bar chart. The loss probability is fixed such that the techniques do not encounter loss jumps within a measurement period.

First, *LC* performs comparably well with all frequencies. The longer the period is, the lower the error. *PR* and *IC* are stable as they are independent of the measurement period, which also means that they cannot force updates at certain times. Nevertheless, the figure reveals that, under the assumption of steady loss rates, *SC* becomes a feasible alternative when the update times are comparably high. *SC* requires large numbers of traversing packets to compensate for the information loss due to sampling.

Discussion of Costs

To compare the techniques within different situations, it requires, despite a comparison of the accuracy, a discussion of the technique's costs. First, most obvious, one should avoid the active probing technique (*PR*) whenever another technique is viable. *PR* introduces notable additional traffic on the pro-

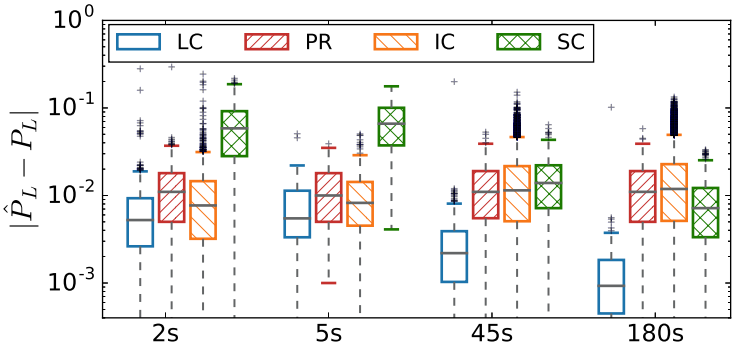


Figure 41: Accuracy with different measurement frequencies. The accuracy of *LC* and *SC* depends on the frequency, whereas *PR* and *IC* do not. If the frequency is low enough, *SC* gathers enough packet samples such that it delivers usable results [Har+17b].

ductive channels such that it disturbs traffic and degrades the performance for the sake of monitoring. Furthermore, one should favor sampling-based counters (*SC*) to legacy counters (*LC*). Although the transmission costs to fetch the counters are equal, switches need less resources to collect counters when using coarse-grained counters (left open their implementation) compared to fine-grained counter. The in-line counter technique (*IC*) does not produce notable transmission costs and the counting costs are equal to *LC*. A comparison of *IC* with *SC* is not easily possible due to the different cost types; however, as fine granular counters, used in *IC* require, cost intensive-TCAM, we assume *SC* to be less resource consuming.

4.1.3 *Adaptive Selection of the most Suitable Technique*

Based on the findings in the previous section, we can decide which technique should be favored depending on the network conditions and monitoring requirements. To represent different conditions, we used changing traffic intensities and types as well as changing loss dynamics. Different measurement frequencies represent different monitoring requirements.

Table 4 summarizes the resulting recommendation. First, if there is no to minimal traffic, given in the first row, the *PR* technique should be preferred independent of the measurement period. As all other techniques rely on production traffic to conduct the estimation for P_L, *PR* is the only sensible choice. If there is friendly (non-bursty) traffic with low intensity, we avoid *PR* and find *SC* not accurate enough. Although *LC*, *IC* (and *PR*) have errors of less than 2% for over 75% of all values (cf. Figure 38, p. 77), *IC* is vulnerable to dynamic loss profiles (cf. e.g., Figure 39, p. 77) such that only *LC* suits these conditions. If we face friendly, yet intense traffic, we distinguish between high and low measurement frequencies. If the frequency is low, *SC* is the least expensive technique, which delivers reasonable accuracy. However, if the frequency

Table 4: Technique selection scheme based on coarse traffic properties and the desired estimation interval.

Traffic Profile	*Short Periods*	*Long Periods*
Minimal	*PR*	*PR*
Friendly, Low	*LC*	*LC*
Friendly, High	*IC* / *LC*$^{\triangle}$	*SC*
Bursty, Low	*LC* / *PR*$^{\circ}$	*LC* / *PR*$^{\circ}$
Bursty, High	*IC* / *LC*$^{\triangle}$	*SC* / *LC*$^{\triangle}$

$^{\triangle}$*LC* for failure detection $^{\circ}$ *PR* if probing traffic acceptable

is high, thus the period is short, *SC* lacks accuracy (cf. Figure 41, p. 80). Therefore, with friendly, intense traffic and short measurement periods, *IC* is of choice assuming that the loss does not excessively fluctuate (otherwise, e.g., to detect loss jumps due to failures *LC* is the better choice). Further on, in row four of Table 4, we consider bursty traffic conditions with low intensity. In both cases, *PR* and *LC* deliver the best results. If probing is acceptable, *PR* should be favored as it does not depend on sporadic traffic. Considering the same conditions but with higher traffic intensity, also *IC* and *SC*, which are less costly, provide reasonable accuracy. *IC* works better with short periods, where *SC* does not see enough packets in defiance of the high traffic intensity for reasonable estimations (cf Figure 41, p. 80). However, if the period is long enough, *SC* is of choice. Both showed to be effective only if the loss probability is comparably steady such that *LC* should be preferred for failure detection.

As a consequence of the technique's varying performance depending on the network conditions and measurement requirements, the monitoring applications should use the concept of transitions [Alt+19] to exchange the used technique. In the following, we show transition-enabled monitoring that switches flexibly between the active technique. For this, in Figure 42, we vary the network conditions and observe the loss estimation accuracy and qualitatively illustrate the costs. In the uppermost subplot, the error $|\hat{P}_L - P_L|$ reflects the current accuracy. The lower the better. The inner subplot shows the traffic intensity, starting from no traffic up to 10^5pps with a step at 10^2pps in the second phase. The lower subplot qualitatively depicts the costs split up in costs for counter collection (dashed orange) within switches, traffic overhead on the data-plane (solid yellow), and traffic on the control-plane (solid red spikes).

Starting with the first phase, the condition that no traffic is within the network requires the monitoring to use *PR* according to our recommendation. The error is almost always below 0,04 in this phase; however, the costs for control- and data-plane traffic overshadows this phase. In the next phase, we change the packet rate to 10^2pps such that *LC* can be used and all data-plane

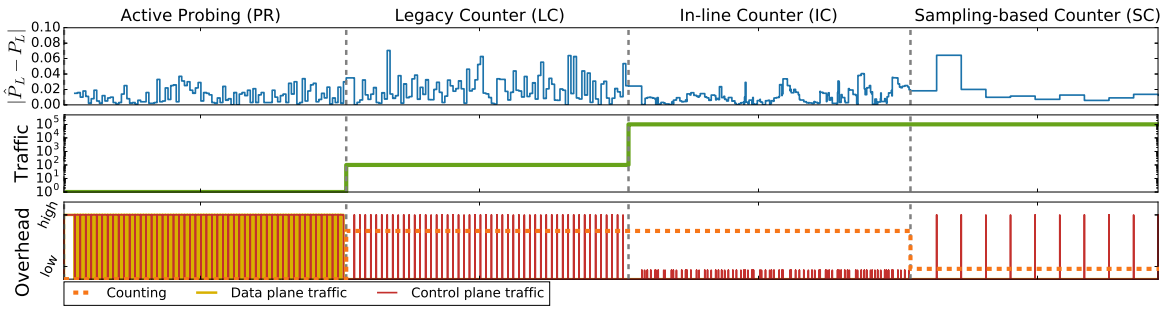


Figure 42: Changing the technique in different phases with different traffic conditions and measurement frequency requirements. With no traffic, the active approach is of choice in the first phase introducing high data-plane and control traffic. With low traffic, legacy counters are of choice (second phase), with high traffic the in-line counter technique (third phase). Both techniques introduce counting overhead; however,different control traffic costs. Only with a lower update frequency and high traffic, the resource friendly sampled counter technique is usable in the last phase [Har+17b].

costs avoided. Relating the costs, we see that now only periodic messages burden the control path to transmit counter. In addition, the overhead for counting increases. The accuracy decreases only little and stays within an error of under 0,04 on average. In the third phase, the traffic intensity further increases to 10^5pps. Based on the given recommendation, the monitoring now uses *IC* and achieves an excellent accuracy. Furthermore, the costs for transmission decrease to a minimum, as *IC* only piggybacks counter information to control messages. As visible, the counter updates are not periodic but sufficiently often. The counting costs remain similar to *LC*. We assume for the last phase that the measurement requirements change and updates are expected only every 45 seconds instead of every two seconds. In combination with high traffic intensity, *SC* is of choice due to minimal cost overhead with reasonable accuracy. The uppermost plot reveals again a good accuracy almost always under 0,02. Furthermore, the counting costs decrease as only coarse-grained counters must be collected and to counter transmission costs come rarely.

4.1.4 *Summary*

In conclusion, we showed in this section that different techniques have a different performance with respect to the environment, thus the conditions, such as the traffic profile, as well as the requirements (*here:* the measurement update frequency). Due to the varying technique's behavior, it is favorable to use transitions between different techniques being aware of the conditions. In an artificial scenario, we show how transition-enabled monitoring would maintain good accuracy while minimizing the costs. Although all techniques can be optimized further to increase their accuracy, no single technique is superior in every condition so that we propose to flexibly exchange it.

4.2 Online Measurement Point Selection

The second treated aspect of the monitoring data collection process is *where* to measure statistical information. To capture the full state of a network, monitoring applications face a tremendously large amount of measurable information. A disproportionality between costs to measure everything and information gain for each individual measurement leads to inefficiency of the monitoring process. Methods such as timely or spatial sampling (cf. Section 2.3) allow reducing this discrepancy by estimating the overall state given only a subset of potential measurements. In this section, we target to select measurement points (spatial sampling) in order to maximize the insights of information they provide. A number of works tackle the very same problem [Cha+05; Sie+14b; Sie+14a]. However, these works almost exclusively formulate the placement as an optimization problem. The computation of solutions for constraint-based optimization problems are known to have excessive runtime [ZH07] and are, therefore, not suitable for dynamic networks where measurement points must potentially be replaced in short time intervals. Many approaches provide heuristics to approximate the optimal solution, which, first, lose the unique characteristics of being optimal and, second, still requires considerable calculation runtime. In contrast, we target a fast calculation of measurement point placement that can be recalculated online. As an exemplary metric of interest, the monitoring application calculates the flow size distribution [TV11], holding information about the distribution of flow lengths in packets. We propose placement strategies under two different assumptions: *(i)* Central knowledge on traffic is available in the controller allowing to leverage information about flows given from the routing application, and *(ii)* without knowledge on traffic, we place measurement points based on the notion that a representative number of flows traverse through *central* nodes in a network. The centrality of a node can be calculated using *centrality scores*, introduced later. This section is based on [Har+18b].

4.2.1 *Measuring the Flow Size Distribution*

Before we describe the placement approach, this section elaborates the measured metric and how it is captured.

Flow Model

Although broadly used, the term *flow* is not defined ultimately. Most definitions denote a flow as a set of packets sharing common characteristics, particularly, common header fields. With that definition, OpenFlow and the alike use flow tables to steer flows through a network. Thus, the shared header fields allow that all packets of a flow are handled similarly within a network part, e.g., all packets with the same source IP and destination IP traverse the same path, or all packets with a specific destination TCP port will be dropped.

A flow might aggregate multiple subflows, e.g., different applications communication between the same endpoints. For the sake of simplicity and clarification, here, we assume that one flow rule processes exactly one flow.

Flow Size Distribution

The flow size distribution (FSD) shows the length of flows within a network. In this work, the length of flows is defined as the number of packets within a single flow. Other definitions consider for instance bytes rather than packets. The FSD has different applications such as inferring the traffic type of flows or anomaly detection: Using the packet characteristics and the flow sizes, an operator can determine the traffic type [Kum+04] (e.g., multimedia content) and provision the network accordingly to maximize the Quality of Service. A supreme example of applications for the FSD is also the detection of Distributed Denial of Service (DDoS) attacks [Kim+04]. Many sources target a single destination with minimally sized flows, e.g., single Ping or TCP SYN packets, to overload the destination. The service becomes unresponsive (*denial of service*). The sudden occurrence of a large number of flows with size one infers such an attack. Moreover, simple worm attacks can be detected as they result in a large number of flows with the very same size [Kim+04].

Tune and Veitch [TV11] provide a formal definition of the FSD, which we adopt in this work. Let N_f be the total number of flows. The i'th flow has the size (packet count) m_i. Now the maximum flow size is $W = \max_i \{m_i\}$ with $W \in \mathbb{N}^*$, therefore, $1 \leq m_i \leq W$. The other way around, M_j denotes the number of flows that have the size j. Now, the total number of flows N_f is then $N_f = \sum_{i=1}^{W} M_i$. The FSD is the set $\theta = \{\theta_1, \ldots, \theta_W\}$ that contains all flow occurrence ratios. A single ratio can be calculated using Equation 17, where θ_j is the number of flow having size j divided by the total number of flows.

$$\theta_j = \frac{M_j}{N_f} = \frac{M_j}{\sum_{i=1}^{W} M_i}. \tag{17}$$

Capture a Flow Size

To capture the size m_i of the i'th flow, OpenFlow provides different possibilities. The intuitive approach is to fetch the packet counter of the corresponding flow entry within a certain time interval and update the FSD accordingly. However, as Figure 43 shows, periodic statistic requests, shown in red, falsify the flow sizes. First, the short flow consisting of four packets is completely missed as the request frequency might be longer than the flow's lifetime. Small flows are underrepresented in such cases. Furthermore, each measurement during the lifetime of a flow will tell the monitoring a false total flow size as more packets are expected. The flow size distribution would be biased towards smaller sizes. Lastly, also shown at the end of the second

flow, the real length might not be captured at all if the flow expires before a new statistic request is sent.

Figure 43 shows a better approach leveraging OpenFlow specific[37] *FlowRemoved* messages (dashed blue), which are sent whenever a flow rule expires. These messages contain statistical information including the total packet count of the expired flow. Using that counter, we solely use correct flow sizes in our FSD, assuming that a single flow rule represents exactly one flow. Yu et al. [Yu+13] presented a bandwidth measurement approach, also using such messages. *FlowRemoved* messages have the advantage that no additional statistic request and response messages are required. The messages can be activated and deactivated in OpenFlow.

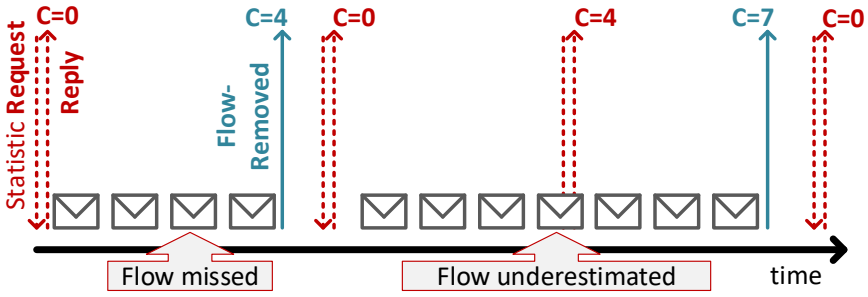


Figure 43: The intuitively way of measuring flows using periodic statistic requests (dashed red) is vulnerable to flow misses and flow underestimation. Inspired by FlowSense [Yu+13], we leverage *FlowRemoved* control messages (solid blue) that contain the correct flow size and are emitted for each flow rule on its removal [Har+18b].

The placement approach supports the selection of multiple measurement points to increase the accuracy. If multiple measurement points (switches) observe the same flows, the data used for the FSD must be cleaned in order to avoid using redundant measurements of the same flow. Usually, flows can be identified through their 5-tuple[38] or OpenFlow's flow entry cookies [Pfa+12]. However, if the controller aggregates multiple subflows into one flow rule, it is hard to compare the contained subflows within one switch with flows seen in other switches. The identification and measurement of subflows within a single flow rule is left for future work.

4.2.2 *Placement of Measurement Points*

In the introduction of this section, we motivated that there is a disproportionality between the information gain and costs when measuring the network's state everywhere. To overcome this, we propose placement mechanisms to

37 It can be assumed that other/future SDN protocols provide similar functionality as it is an essential requirement for many management applications, including routing.

38 The 5-tuple identifies TCP/IP and similar connections: `(Source IP, Destination IP, Protocol, Source port, Destination port)`. Note that *flows* can be organized differently.

collect as much information as possible using small numbers of measurement points. The mechanisms can easily be used for rapid replacements as they depend only on simple calculations.

Measurement Point Selection with Knowledge on Flows

The first considered scenario, with the assumption that knowledge about flows in the network is available, e.g., through communication with the routing application, allows a straight forward placement to maximize the information gain with respect to the flow size distribution. To have the most representative measurement point, we select the switch that sees the highest number of flows in the network. We model the network as a graph $G(V, E, F)$ consisting of nodes/vertices (switches, potential measurement points) $V = \{1, 2, \ldots, N\}$, edges $E = \{(v_i, v_j) \mid i, j \in V, i \neq j\}$, which are links between nodes, and flows $F = \{f_1, \ldots, f_{N_f}\}$ with N_f being the total number of flows. A flow can be expressed as a sequence of vertices: $f_l = \{i_1, i_2, \ldots, i_k\}$ while $\{i_1, i_2, \ldots, i_k\} \subseteq V$. The number of flows on the switches is $C(i, F) = \sum_{f_l \in F} \mathbb{1}(i \in f_l)$ for $i \in V$. Given that, we select the switch with the highest number of flows traversing through it with

$$s_1 = \arg\max_{i \in V} C(i, F). \tag{18}$$

Figure 44 shows exemplary that the switch s_1 (marked with s_1) will be selected as measurement point since it sees three flows, whereas all other switches see less.

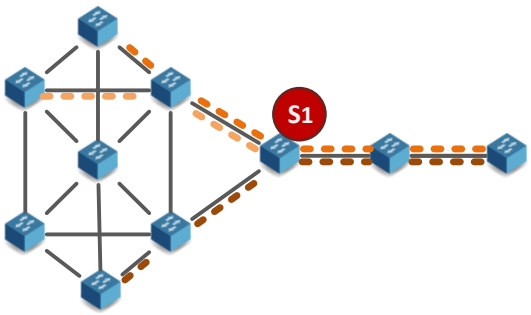


Figure 44: Selecting the switch based on information on flow paths. The marked switch sees three flows, which is the highest number in this example and, therefore, provides the most information [Har+18b].

If more resources are available and further measurement points can be selected, the same method is used redundantly. However, when calculating the number of flows traversing through a switch candidate s_{i+1}, we ignore all flows captured with any of the already selected measurement points, namely

$\{s_1, s_2, \ldots, s_i\}$. Say $F_{s_i} = \{f_l \in F : s_i \in f_l\}$ is the set of flows traversing through a switch s_i. Then, the calculation of the number of flows per switch alters to

$$s_{i+1} = \arg\max_{j \in V} C(j, F \setminus \cup_{k=1}^{i} F_{s_k}). \tag{19}$$

Measurement Point Selection without Knowledge on Flows

The second investigated scenario assumes that the monitoring application cannot access information on flow paths. Hence, to find the switch that provides the highest amount of information, we follow the notion that *central* nodes in a network are of more importance for the overall network and, therefore, provide the most information. To find *central* nodes, the topology serves as basis. In other network management applications, centrality metrics from the social network theory [EB99; Bor06] proved to be useful [Cuz+12; Cha+17]. We borrow such scores to place measurement points fast in vital positions, which is novel to the field of monitor placement. Solely Yoon et al. [Yoo+17] used centrality metrics to pick important nodes for an intrusion detection system. Although a large number of possible centrality scores exist, we limit to the ones that can be calculated fast[39] such that they are suitable for the desired online placement. Figure 45 depicts different selections based on three different used centrality scores, which are described in the following.

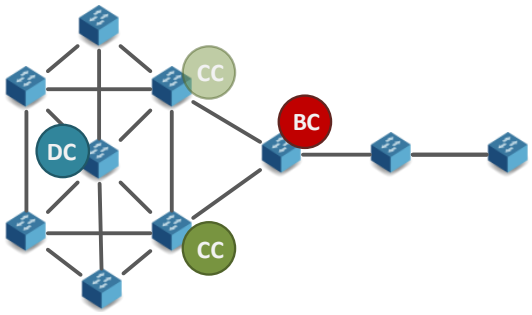


Figure 45: Measurement point selection without knowledge on flows considers only the topology. Centrality scores give estimates about the importance of a node. The figure shows nodes with the highest betweenness centrality (BC), closeness centrality (CC), and degree centrality (DC) [Har+18b].

Betweenness Centrality [Fre78]: The betweenness centrality rates nodes based on the number of shortest paths they are part of. Particularly in networks with shortest-path routing, this metric is a good candidate to select measurement points according to their relevance and the information they provide on flows. In Figure 45, BC marks the node with

39 During the implementation of a prototypical system using centrality scores, the eigenvector centrality [R00] showed poor performance and was dropped as candidate.

the highest betweenness centrality score. Say the number of shortest paths between a node $i \in V$ and $j \in V$ is $\rho_{i,j}$. Furthermore, the number of shortest paths between i and j traversing through a particular node $k \in V$ is $\rho_{i,j}(k)$. The normalized betweenness centrality $BC(k)$ of any node k is given with

$$BC(k) = \frac{2}{(|V|-1)(|V|-2)} \sum_{\substack{i,j \in V \\ i \neq j \neq k}} \frac{\rho_{i,j}(k)}{\rho_{i,j}}. \tag{20}$$

Closeness Centrality [Fre78]: The closeness centrality reflects the inverse distance to all other nodes in the network. Hence, a node with a high closeness centrality score is located in the core of the network, which plays a major role in many communication network topology structures. Such nodes spread information efficiently throughout the network. In the figure, two candidates, marked with CC, have the same closeness centrality. One of the nodes will be selected randomly. Let $d(i,j)$ be the distance, e.g., hop count, between nodes i and j. The closeness centrality $CC(k)$ of a node k is then

$$CC(k) = \frac{|V|-1}{\sum_{i \in V, i \neq k} d(i,k)}. \tag{21}$$

Degree Centrality: The degree centrality takes the number of connections to other nodes into consideration. Therefore, the representativeness is limited and strongly depends on the topology's structure. The node with the highest number of connections is marked with DC in the figure. Say $a(i,j) = 1$ *if* $(i,j) \in E; 0$ *otherwise*, thus a connection exists between i and j. Then the degree centrality $DC(k)$ for a node k is

$$DC(k) = \sum_{i \in V} a(i,k). \tag{22}$$

By leveraging the presented centrality scores from the social network theory, the selection of the first measurement point s_1 is straight forward. However, a typical characteristic of these scores is that close-by nodes often have similar scores, e.g., nodes in the core of the network have similar closeness centrality etc. Unfortunately, the set of flows traversing through close-by nodes also overlaps heavily. As a consequence, simply select the node with the second highest centrality score as the second measurement point leads to poor efficiency. To overcome this, we constraint the selection of additional measurement points while still considering the centrality scores. Intuitively, we target to consider only candidate nodes if the ratio of shortest paths between the already selected nodes and the new candidate differs significantly. Say the set of shortest paths between two nodes j and k that pass through a node s_i is $P_{j,k}(s_i)$. Vice versa, the shortest paths that flow through that

node s_i is $P(s_i) = \bigcup_{j,k \in V} P_{j,k}(s_i)$. Before considering candidate s_{i+1} as the next measurement point s_{next}, we make sure that Inequality 23 is fulfilled.

$$\frac{|P(s_{next}) \setminus \bigcup_{j=1}^{i} P(s_j)|}{|\bigcup_{j=1}^{i} P(s_j)|} \geq \delta_{min}. \tag{23}$$

If not, we try nodes $s_{next} = s_{i+2}, s_{i+3}, \ldots$ ordered ascendingly by their centrality score. δ_{min} defines the minimum portion of shortest paths that must be disjunct for the candidate. By default, we set $\delta_{min} = 1/2$, so at least 50% of the shortest paths passing through the candidate must not be captured with already selected measurement points. If no candidate is suitable, we select the one with the largest number of disjoint shortest paths.

4.2.3 *Evaluation*

In this section, we evaluate the measurement point selection and discuss its effect on the relationship between costs and information gain.

Evaluation Environment and Methodology

To conduct the evaluation, we set up a virtual MININET [LHM10] network. As the placement of measurement points strongly depends on the topology, each metric was captured within different topologies taken from the Internet Topology Zoo [Kni+11] plus a data center topology [Roy+15] as described in Section 3.3. The majorly shown *Surfnet* topology is meshed, while the data center topology follows a tree structure. Other topologies are detailed when used and are detailed in Section A.7. The Ryu OpenFlow controller[40] serves as management unit with a monitoring application that implemented the flow size distribution measurements. Furthermore, the controller provides shortest-path routing. If not stated differently, in each evaluation run, we send 100 flows from a randomly chosen source to a randomly chosen destination. Hosts are by default connected to each switch within the topologies with the exception of the data center, where only the racks are hosts (cf. Section 3.3.1). Traffic is sent using RUDE/CRUDE[41] with exponentially distribution packet interarrival times and constant bit rate. To model flow sizes/length in bytes, we pick a random number from a Zipf distribution with parameter $s = 1,6$ such that we have majorly short *mice* flows and rare long *elephant* flows.

Each metric is captured with at least 30 repetitions. We majorly focus on the accuracy and costs to assess the efficiency. First, we determine the *accuracy* using the *Bhattacharyya distance* [Bha46]. This distance is a measure for the similarity of two distributions f and g between 0 and ∞. The closer it is

40 Ryu SDN Framework `https://osrg.github.io/ryu`, accessed 16 May 2019

41 Real-time UDP Data Emitter (RUDE) & Collector for RUDE (CRUDE) `rude.sourceforge.net/`, accessed 16 May 2019

to zero, the less distance between the distributions. First, the *Bhattacharyya coefficient* $\rho(f, g)$ gives an estimation of the overlap.

$$\rho(f, g) = \sum_{x \in X} \sqrt{(f(x) \cdot g(x))} \tag{24}$$

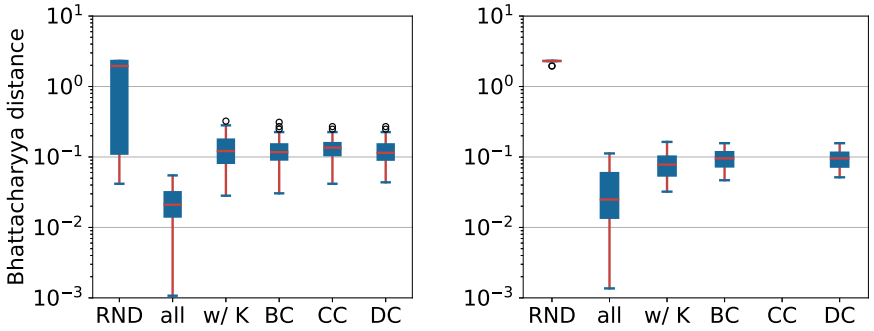
Using the coefficient, the distance $B(f, g)$ is calculated using

$$B(f, g) = -\ln(\rho(f, g)). \tag{25}$$

In addition, we determine the *costs* for the measurements by counting the required bytes to collect the statistics from the measurement points.

Accuracy and Cost Comparison with Different Placement Methods

First, this section demonstrates how the different placement methods perform in terms of achieved accuracy and required costs. For this, we compare the placement methods for a single measurement point with each other, with a random measurement point placement (*RND*), and with using all switches as measurement points (*all*). The abbreviation *w/ K* translates to *with knowledge about flows*, *BC* to betweenness centrality, *CC* to closeness centrality, and *DC* to degree centrality.

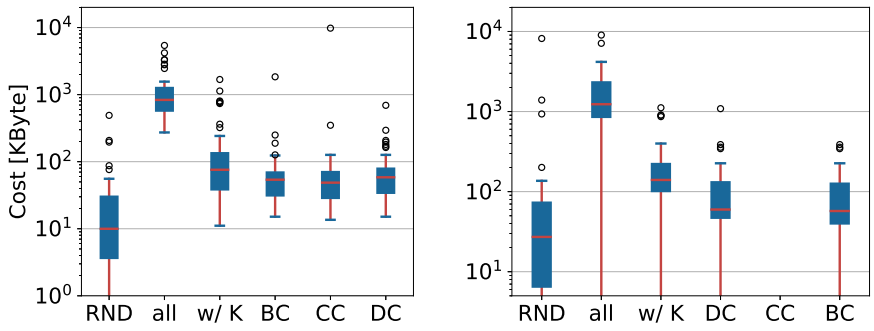


(a) Error in the Surfnet topology. (b) Error in a data center topology.

Figure 46: Accuracy (shown as error; the lower the better) for different measurement point placement strategies with only one measurement point. The results indicate that the placement strategies are sensible and, therefore, significantly better than a random placement (*RND*). Measuring everywhere (*all*) has the best accuracy. There is no distinct difference between the selection strategies [Har+16].

Figure 46a (Surfnet topology) and 46b (data center topology) show the accuracy comparison of the aforementioned techniques. Note the logarithmic y-axis. In both topologies, using all measurement points (*all*) leads to the

lowest distance between the true distribution and the estimated distribution, thus the best accuracy as all flows are covered. Looking at the placement strategies, either with knowledge or without knowledge, all show similar results that cannot be distinguished easily. Surprisingly, the knowledge about flow information does not significantly improve the accuracy compared to centrality-based placements. However, the selection strategies are all sensible as the comparison with a random placement (*RND*) provides much worse results for all topologies. Also, note the missing box for the closeness centrality within the data center network. The closeness centrality selects one of the root switches of the tree and all hosts are leaves. The shortest-path routing uses one of the two root switches while the other one is ignored. The placement using betweenness centrality selects the root switch that does not see any flows. Although this topology-dependent case is very specific, it shows that a careful consideration of the topology is required when choosing the placement strategy.



(a) Costs in the Surfnet topology. (b) Costs in a data center topology.

Figure 47: Measurement costs for different measurement point placement strategies with only one measurement point. The costs are inverse to the accuracy shown in Figure 46. Using centrality-based placement strategies is less costly than measuring with respect to the total number of captured flows (*w/ K*) [Har+16].

Furthermore investigating the costs, shown in Figure 47, first of all, we observe that using all measurement points produces significantly more statistic transmission overhead than the other techniques (again, note the logarithmic y-axis). Furthermore, the random measurement point selection produces the least costs. This is due to the fact that the random point sees fewer flows than a measurement point in a central position. This reasons also the poor accuracy of a random selection. Considering our proposed selection strategies, we see that, by analogy to the accuracy, we face almost equal costs. Nevertheless, a slight cost reduction for centrality-based approaches is visible, particularly looking at the median of *w/ K*, *DC*, and *BC* of Figure 47b. As the selection based on the number of flows maximizes the number of measured flows, this produces the highest costs for statistic transmissions consecutively. Relating

that with the accuracy, we find that the measurement point selection only based on the topology (centrality) is comparably accurate as of the flow-based selection, whereas they reduce slightly less costs.

Accuracy within Different Topologies

In the next evaluation, we compared the accuracy of the selection strategies in different topologies. Figure 48 shows the closeness centrality representative for all centrality metrics. Other strategies produced similar results and, therefore, did not deliver further insights. The data center topology is colored blue (first), topologies that follow a tree or star structure are colored green (second to fifth), and orange boxes show meshed topologies (remaining three), including the previously known Surfnet topology. Appendix A.7 shows details of the topologies.

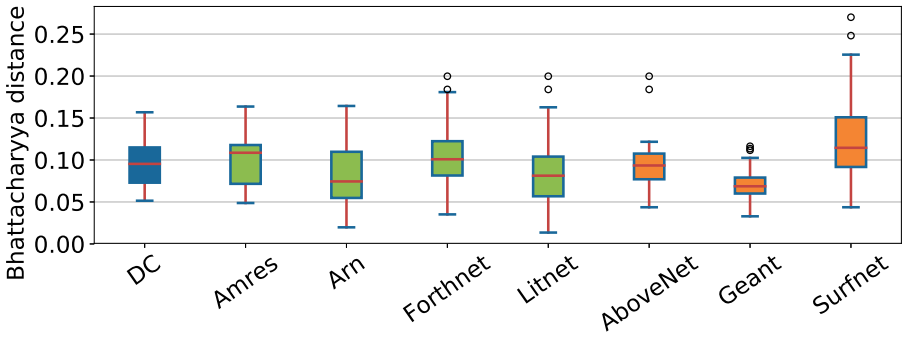
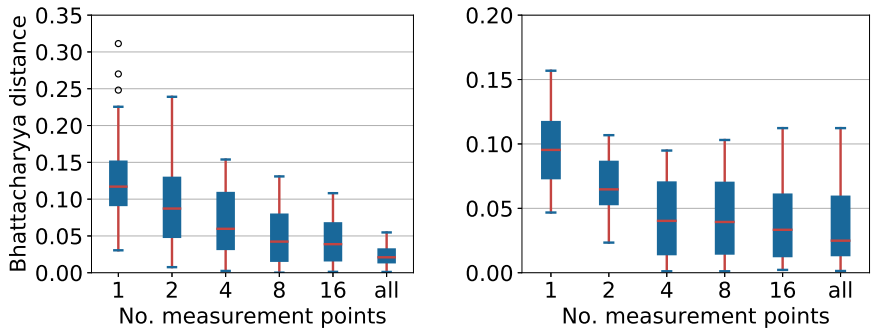


Figure 48: Accuracy (shown as error) for different types of topologies using the closeness centrality as placement strategy for a single measurement point. The blue box represents a data center topology, green boxes show tree/star-like topologies, and orange boxes show meshed topologies. No distinct dependence on the topology structure is observable [Har+18b].

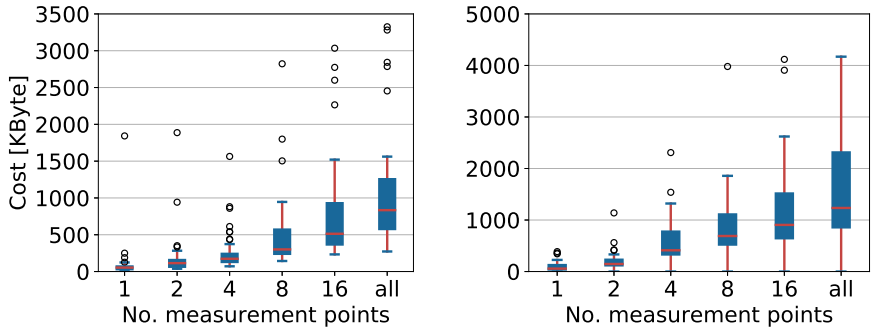
The figure reveals that the centrality-based selection does not favor any specific topology. Although the accuracies differ slightly, there is no clear pattern. However, in our evaluation, the flow size distribution was equally applied among the whole network such that the number of captured flows is crucial for the accuracy. Yoon et al. [Yoo+17] argue that traffic potentially looks different depending on the location within a topology. For instance, the edge network parts see other types of traffic than the core. We argue that the placement strategies with multiple measurement points make sure that the observed shortest paths are majorly disjoint and, therefore, with a certain number of measurement points, traffic everywhere in the network will be evenly observed. We leave the investigation and further improvements to observe all parts of the network for future work.

Accuracy and Cost Comparison using Multiple Measurement Points

This section contains results on the accuracy and costs when increasing the number of measurement points. Figure 49 shows the accuracy as *Bhattacharyya distance* (upper subfigures) and costs in bytes (lower subfigures) on the y-axes with different numbers of measurement points on all x-axes. Figure 49a shows accuracy results for the Surfnet topology. It is visible that the median distance decreases with more measurement points. In addition, the whiskers become smaller, indicating smaller variance when using more measurement points. For the data center topology, depicted in Figure 49b, we observe that starting from approximately four measurement points, the accuracy does not improve further. Most likely, within the data center with shortest-path routing, a small number of measurement points covers almost all flows.



(a) Error in the Surfnet topology. (b) Error in a data center topology.



(c) Costs in the Surfnet topology. (d) Costs in a data center topology.

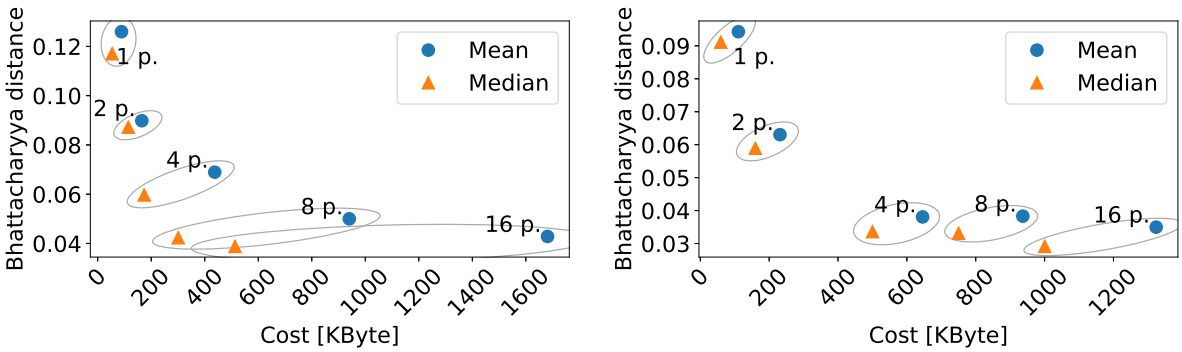
Figure 49: Accuracy (shown as error) and costs measurements for different numbers of measurement points using the betweenness centrality-based placement. The figures reveal that the accuracy increases for both topologies; yet, in the data center topology it saturates after four measurement points. The costs increase steadily with the number of measurement points [Har+16].

In contrast to this, Subfigures c and d of Figure 49 depict the costs with an increasing number of measurement points. For both topologies, we can see a steady increase of the costs for statistic transmission when measuring at

more switches. In both figures, the whisker size increases as well, meaning that often additional measurement points do not cover many flows whereas sometimes they do. Complementary to the accuracy, the costs do not saturate in the data center topology with more than four measurement points.

Information Gain per Measurement Point

The previous results already indicate that the relation between accuracy improvement and costs is not equal for each measurement point. The following results confirm that result and show particularly that an intelligent measurement point selection sharply increases the efficiency.



(a) Accuracy to cost relation in the Surfnet topology.

(b) Accuracy to cost relation in the data center topology.

Figure 50: The figures compare the error with the costs using betweenness centrality-based placement and different numbers of measurement points. Both figures show a definite disproportionality between accuracy and costs when many measurement points are used: After two and four points, respectively, the accuracy does not increase, however, the costs increase [Har+16].

Figure 50 shows the mean and median accuracy (y-axes) and the corresponding mean and median costs (x-axes) when using different numbers of measurement points. Subfigure a shows results using the Surfnet topology and Subfigure b results using the data center topology. Both show consistently that the accuracy is the lowest when only one or two measurement points are used (large *Bhattacharyya distance*). Yet, for both cases, the low number of measurement points leads to the least costs. Increasing the number of measurement points increases the costs on the x-axes, also coherently among both figures. However, using more than four measurement points within the Surfnet topology and more than two measurement points in the data center does not optimize the mean and median accuracy further. In both scenarios, after a certain threshold, additional measurement points do not enhance the quality. The accuracy remains the same, whereas only the costs increase. This supports the motivational hypothesis from the beginning of this section that not all measurement points deliver a comparable amount

of relevant information. Therefore, we propose the sensible selection of measurement point locations to have the most information with the first new measurement locations. The figures reveal that the first measurement points contain most information content.

4.2.4 *Summary*

In this section, we investigated the question of *where* to measure statistical information. With the notion that not all measurement points provide equally important information, we proposed placement strategies to maximize the information captured with small numbers of measurement points. The strategies base on knowledge about flows and centrality scores to estimate a node's informativeness, respectively. In the evaluation, we could confirm the notion and show that an intelligent placement of measurement points allows gathering most information available in the network with comparably few measurement locations for the example of a flow size distribution estimation. Additionally used measurement points only add costs, whereas they do remove uncertainty from the estimation.

4.3 Application-driven Selection of Measurements

The third and last aspect within the monitoring data collection process tackles the question *which* statistical information to measure. The term statistical information refers to measurement values rather than the metric as the latter is fixed based on the management application requirements. Since it is not viable to capture every statistic reflecting the network state [KCG14; HG12; Gio+14], we tackle the superior goal of increasing the efficiency by maximizing the information gain per measurement. As already shown in the previous section, not all measurements provide a reasonable amount of relevant information with respect to their costs. Measurements produce threefold costs: *(i)* Processing overhead to capture the metric in the data-plane; *(ii)* transmission costs from the data- to the control-plane; and *(iii)* processing costs within monitoring applications, which use primitive metrics to derive the complex network state. We propose to only forward measurements to the control-plane that contain information not yet present in controllers and that significantly improves the monitoring quality in order to save costs. Mainly, we target to save costs for transmission and processing on the control-plane. To this end, we use an intermediate layer between the control- and data-plane to support the measurement process (integrates into the architecture from Section 3.1). The entity estimates the importance of measurements based on their history. Subsequently, we extend the work to avoid a centralized entity by pushing the preprocessing functionality into the data-plane. To attain that, we leverage programmable data-planes, particularly P4 [Bos+14], to estimate the im-

portance of a measurement based on its use for the consuming monitoring application. Finally, we provide an evaluation of both variants.

4.3.1 *Filtering Between the Data- and Control-Plane*

In a first step, which is based on parts of [Har+18a], we filter irrelevant measurements within an intermediate entity between the control- and data-plane. Figure 51 depicts the architecture.

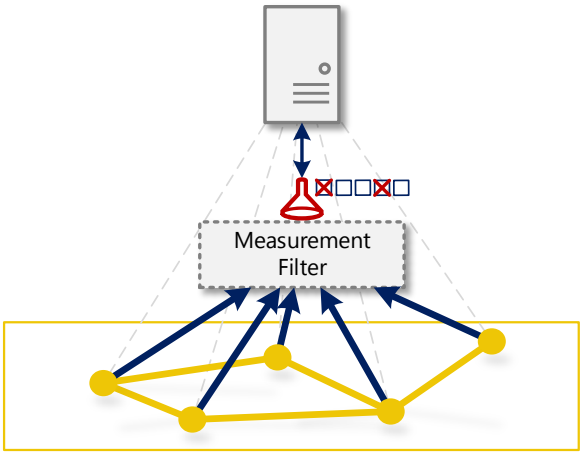


Figure 51: The *Coordination* entity performs, despite aggregation of monitoring tasks (cf. Section 3.1), the filtering of measurements between the data- and control-plane.

The entity entitled *Measurement Filter* (in the following only *filter*) is the very same entity that aggregates monitoring tasks of multiple controllers, denoted *Coordinator*, known from Section 3.1. Therefore, it conducts the measurements on behalf of the controllers. The switches answer with statistic responses to the filter, which forwards them to the interested controllers. Before that, the entity preprocesses and filters the measurements.

Filter Component Logic

Figure 52 describes the components that perform the filtering in the entity. As known from the previous chapter, the component takes monitoring tasks from controllers through its northbound service API (*Task Configuration API*). The *Task Manager* stores, aggregates, and executes the tasks using subcomponents grayed out and the southbound agents. The aggregation functionality was described earlier in Section 3.1.

After a measurement execution, the corresponding southbound agents provide the measurement value to the *Preprocessor*, as shown in the figure. Here, the entity can derive some higher information metrics, such as the band-

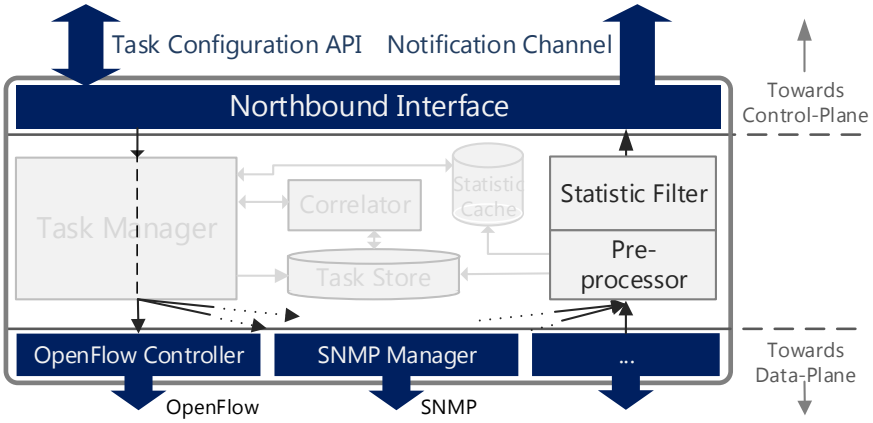


Figure 52: The filtering steps come into picture after the southbound agents captured the primitive metric. First, a *Preprocessor* derives an extended metric if desired, which is then filtered based on the measurement history or an absolute threshold [Har+18a].

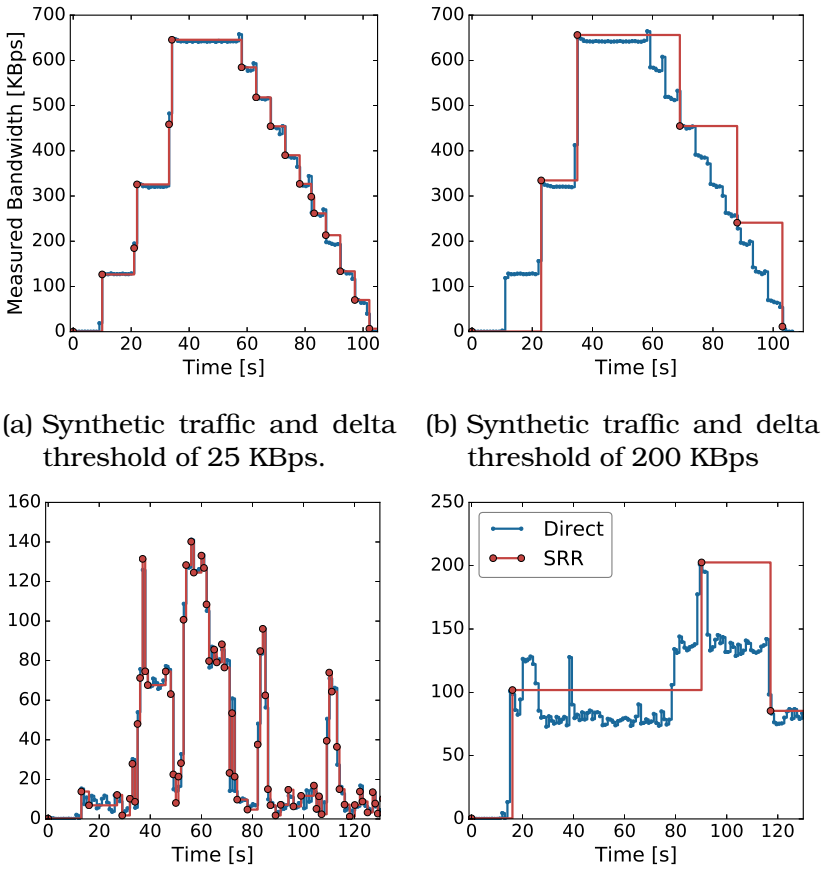
width based on two consecutive byte counter values etc. Afterwards, the *Preprocessor* forwards the potentially derived metric to the *Statistic Filter*. The reason to preprocess and derive primitive metrics before filtering is to allow filtering also on more expressive metrics, which are predominantly used by controllers. During a task registration, the controller can configure different filtering options applied in the filter subcomponent.

Threshold-based: The filter forwards measurement values only if they exceed a certain absolute threshold. For example, it blocks all loss measurements for a link if they are below a particular value.

Delta-based: It filters values, which are close to the previously measured value (cf. A-GAP [PS06]). Only significant changes that are more than a "delta threshold" different from the previously transmitted bandwidth are used. Otherwise the controller works with the old measurements, which are assumed as accurate enough.

Hybrid: After a certain absolute threshold, it dispatches only significant measurement changes. For instance, if the bandwidth is constantly low, there is no need to react in the management applications. However, once it reaches a certain height, the application wants to keep track of it to prepare adaptation, however, only when it significantly changes.

More sophisticated filtering options based on the measurement history are also possible [LC06]. If the filter decides that a measurement is not relevant for the controllers, it is stored to compare future measurement values in the filter. Otherwise, the northbound notification channel transmits the measurement update to the interested controller(s).



(a) Synthetic traffic and delta threshold of 25 KBps.

(b) Synthetic traffic and delta threshold of 200 KBps

(c) Data center traffic and delta threshold of 5 KBps.

(d) Data center traffic and delta threshold of 10 KBps.

Figure 53: Exemplary illustration of the measured network state in the controller when filtering measurements with a small difference to the previous measurements (delta threshold).

Figure 53 shows in two examples how the filtering influences the accuracy in the controllers. The two uppermost subfigures depict the measurement of traffic, which follows step-function. With a low delta threshold of 25 KBps (Figure 53a), the state in the controller (red) follows the state available in the filtering entity (blue) closely. With a higher delta threshold of 200 KBps (Figure 53b), the controller's state diverges clearly from the measurements before filtering. However, first, the state diverges only to the extent the operator specified in the monitoring task configuration. Second, costs for transmission and processing are only caused on every change in the figure. Later, the evaluation will show how this trade-off increases the efficiency of the monitoring. The lowermost subfigures, c and d, show the same investigation for a realistic traffic profile. We observe the same behavior again: Considering a small threshold, the red curve, depicting the filtered bandwidth measure-

ments, strictly follows the blue, directly measured bandwidth curve. However, with a larger threshold of 10 KBps, we have only a rough estimation of the actual bandwidth when filtering. Note that the influence of the threshold strongly depends on the overall bandwidth. In the uppermost two subfigures, where the bandwidth has peaks up to 650 KBps, the thresholds are a multitude higher than in the lowermost subfigures, where the figures face bandwidth up to 200 KBps.

Filtering Workflow

The workflow, shown in the following, is an extension of the aggregation workflow of Figure 16, p. 39. The task registration, storing, and aggregation visible in the referenced figure is neglected in the following. When registering a task, the controllers define filter preferences. E.g., Lines 12 and 13 of Listing 1, p. 36, which say `"threshold": 10000, "threshold_type": "delta"`, define that in the example, all values within the range of 10 KBps around the last measurement should be filtered out. Given a registered task with enabled filtering, Figure 54 shows the filtering workflow.

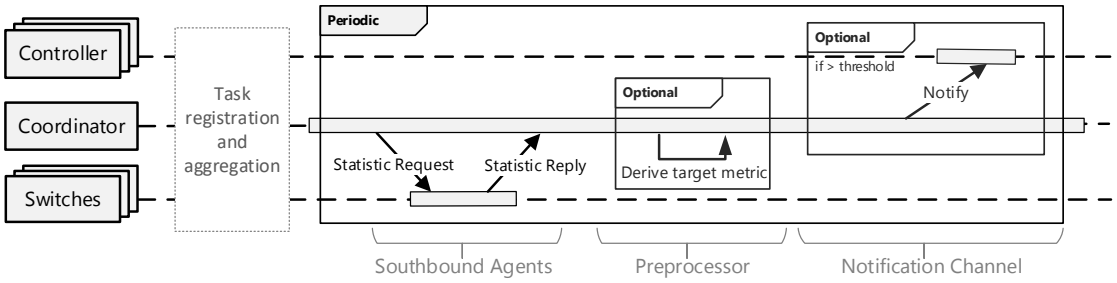


Figure 54: The filtering workflow includes calculating a derived metric based on measured primitive metrics and the filtering based on the task specification [Har+18a].

As shown in the figure, the entity uses one of the agents to request statistics from the data-plane. The data-plane switches answer with the corresponding statistic replies. If desired, the *Preprocessor* processes these value to derive extended metrics, such as the bandwidth instead of byte counter. The resulting value is then compared either against an absolute threshold, the previous values, or both to decide whether it is worth forwarding.

The workflow describes how the intermediate entity prevents costs on the control channel towards the controllers and processing costs on the controllers based on the informative content of a measured value.

4.3.2 *Filtering on the Data-Plane using P4*

The centralized filtering of measurements comes with the same problem as the centralized aggregation mechanism (Section 3.1). The entity introduces a single point of failure, which faces scalability and fail-safety issues, or introduces considerable state exchange overhead when implemented distributively. Furthermore, the measurement transmission costs cannot be suppressed between the data-plane and the filtering entity. A logic step towards an applicable solution is to shift the filtering functionality out of the entity. To prevent unnecessary transmission costs, we target to filter measurement with low information gain in the earliest possible stage.

Based on [Har+19a], in this section, we propose a novel approach for programmable data-planes to filter measurements. Programmable data-planes have already been proven useful in the context of monitoring in SDNs [Siv+17; PAM17; Yan+18; HLB18] as described in Section 2.3; however,restricted to metric capturing within a switch and, to the best of our knowledge, not to distinguish between relevant and irrelevant information, or in combination with machine learning. Figure 55 shows the architecture of the extension. The switches that are queried for statistics check if a statistic is of importance and if not, does not send a reply.

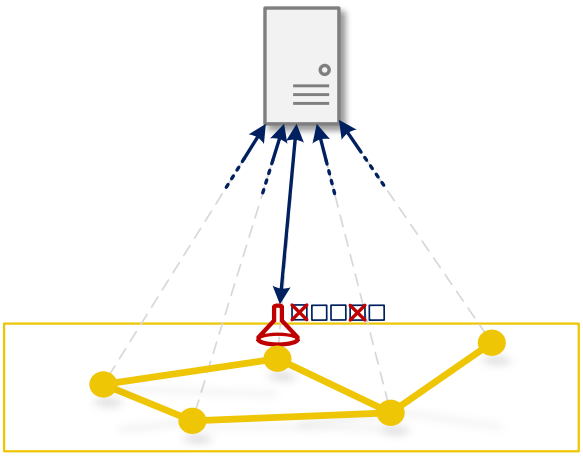


Figure 55: The filter functionality is shifted to the switches. Programmable switches allow customizing the functionality such that the estimation of whether a measurement is of relevance for an application or not can be taken before burdening any networking or computational resources [Har+19a].

In contrast to the latter filtering approach, we furthermore extend it to be application-aware. Instead of filtering based on the history of measurements only, we consider the relevance of each measurement for the application and use machine learning mechanisms to estimate this relevance.

Learning the Relevance of a Measurement

To learn the relevance of a measurement, we require a definition of *relevance*. Generally spoken, the relevance of a particular measurement is the *improvement* of the application's quality considering the measurement compared to the quality without considering it. Although the general concept is designed for any application, many aspects of the machine learning, such as the feature selection, the learning technique, and also the calculation of the relevance/improvement, depend on the applied use case. In the following, we exemplary use a bandwidth forecast application. It predicts the future bandwidth of an investigated link based on an *Auto Regressive Integrated Moving Average (ARIMA)* [Sab77] using prior bandwidth measurements.

First, we define the application's quality improvement metric to assess the importance of a measurement for the application in Equation 26.

$$i = |\underbrace{|bw_{true} - bw_{fc_{n-1}}|}_{\text{last filtered}} - \underbrace{|bw_{true} - bw_{fc_n}|}_{\text{last considered}}| \cdot 1Bps^{-1} \tag{26}$$

Intuitively, it describes the difference between the forecast's accuracy when the latest measurement is included compared to the accuracy when the latest measurement was skipped. The first part of the equation, annotated with *last filtered*, represents the accuracy (shown as error or absolute difference) between the true bandwidth (bw_{true}) and the second last prediction ($bw_{fc_{n-1}}$). The second part of the equation, with the annotation *last considered*, represents the accuracy (also as error/difference) between the true bandwidth and the current prediction (bw_{fc_n}). The subtraction is close to zero if both predictions are comparably good. However, if the error is much smaller when the last measurement was not filtered ($|bw_{true} - bw_{fc_n}| << |bw_{true} - bw_{fc_{n-1}}|$), the subtraction and consequently, the improvement is higher[42]. Thus, i is high if using the measurement decreases the error. Dividing the equation with 1Bps eliminates the unit from the improvement for normalization.

Next, to estimate the improvement before actually having to calculate the forecast, we have to find *features* (learning input parameters) that influence the improvement and are available without calculating the forecast. We base the selection of features mainly on the notion that the forecasts accuracy changes depending on the dynamics of the bandwidths. For example, consider the bandwidth is stable for a couple of measurements. If the next measurement is close to the previous, the forecast will not improve. Nevertheless, if the next measurement significantly differs from the former state (measurement history), the forecast likely improves using the measurement. Out of a

42 The opposing case that the old prediction has a lower error than the new prediction is only of theoretical importance. Such a case occurs if the latest measurement is misleading (e.g., the bandwidth had a short peak and resumes close to its previous amplitude). We take the absolute of the difference so that i is always positive, which suggests an improvement and, therefore, conservatively not to filter the measurement.

pool of 12 features representing the dynamics of the bandwidth, we exemplarily select four features with the highest correlation with the improvement in preliminary tests.

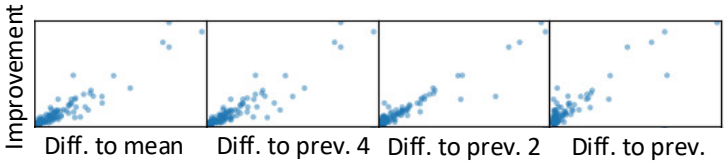


Figure 56: The features (x-axes) that represent the dynamics of the bandwidth history linearly influence the improvement of the forecasting application (y-axis) [Har+19a].

Figure 56 qualitatively shows the four features (x-axes) and their influence on the improvement (y-axis): The first feature (*Diff. to mean*) is the difference of the new measurement to the mean of the previous four measurements; The second, third, and last feature (*Diff. to prev. X*) reflect the sum of differences between the new measurement and the last four, two, and only the very last measurement, respectively. Although the selected features strongly correlate with each other, each gives valuable information to the learner. The figure shows a distinguish linear dependence of the improvement on the features. Therefore, a linear multivariate regression is most suitable to learn the improvement. A linear regression is a basic learning procedure based on Equation 27.

$$\hat{i} = \theta^T \cdot \mathbf{f} = \theta_0 \underbrace{f_0}_{=1} + \theta_1 f_1 + \ldots + \theta_n f_n. \tag{27}$$

The given formula estimates the improvement $\hat{i}$ using the feature vector $\mathbf{f} = [1\ f_1\ f_2 \ldots f_n]$ and the parameter vector $\theta = [\theta_0\ \theta_1 \ldots \theta_n]$. θ_0 is denoted the bias term and $\theta_1\ \ldots\ \theta_n$ as feature weights. As the features can always be calculated using the newest measurement, $\mathbf{f}$ is well-known during runtime. θ, the parameter vector must be learned during a training phase. With a set of training data consisting of improvements i and featured $\mathbf{f}$, we can calculate θ by minimizing the Root Mean Square Error (RMSE) or Mean Square Error (MSE)[43] of Equation 27 [Ger17]. Subsequently, once the parameter vector is found, we can estimate the improvement only with the features to decide whether to send them to the controller or not.

Design of the Data-Plane Filter

The learning phase of the linear regression is solely performed in the controller. For this, the controller deactivates the filtering such that the switch

43 Minimizing a function also minimizes the square of the function and vice versa.

sends every measurement to update the forecast. The controller uses this information to find the improvement for each value using Equation 26 and the measurements to calculate the described features. With that information, it derives the bias term and feature weights (parameter vector). Once this phase is over, the controller changes the executed action for statistic requests to enable filtering. In compliance with Figure 57, the controller pushes the parameters to the switch, which uses them subsequently to estimate the improvement of new measurements. Despite the features that are required to estimate the improvement, the controller stores an improvement threshold. If the estimated improvement exceeds that threshold, the switch sends the measurement to the controller.

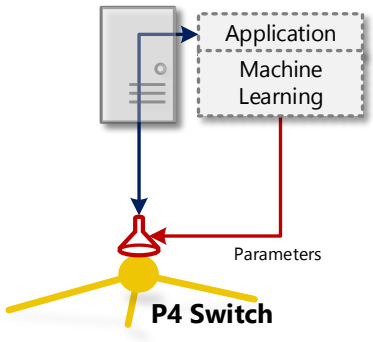


Figure 57: Design of the filtering approach: The controller learns the parameters to estimate the applications quality improvement using training data and pushes them to the programmable switch. Thereafter, the switch filters measurements based on the estimated improvement [Har+19a].

To implement the filtering of measurement within the data-plane, we need to add custom functionality to the switch. Legacy switches and OpenFlow switches provide a predefined set of configurable functions. Therefore, custom functionality cannot be added in a generic, reusable way without changing the switches underlying implementation. As an alternative, programmable switches have been proposed. In the recent years, P4 became the de-facto standard (cf. Section 2.1.3). We leverage P4 to add functionality to a switch to process statistic requests from the control-plane, calculate the linear regression within the switch to decide whether the statistic is worth sending to the controller, and send or drop it. However, calling programmable actions cannot be done for packets entering through the switches control channel, such as statistic requests.

As Figure 58 depicts, the control logic of a switch (hashed red) is vendor-implemented and has, therefore, fixed behavior. As P4 allows programming the switch's behavior, including parsing, processing, and deparsing, for packets that come through a data-plane port, we require the monitoring application to request statistics through a dedicated port (red arrow). After the

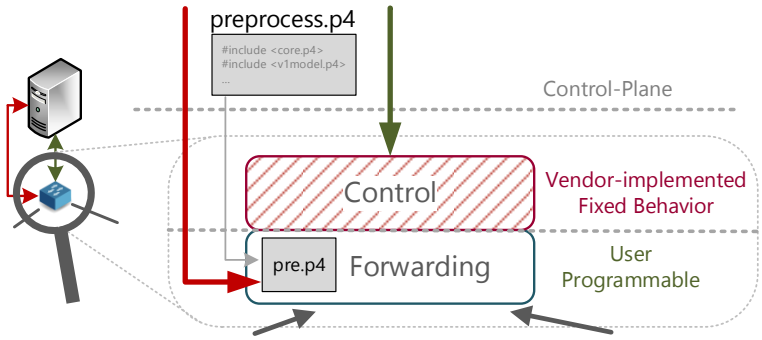
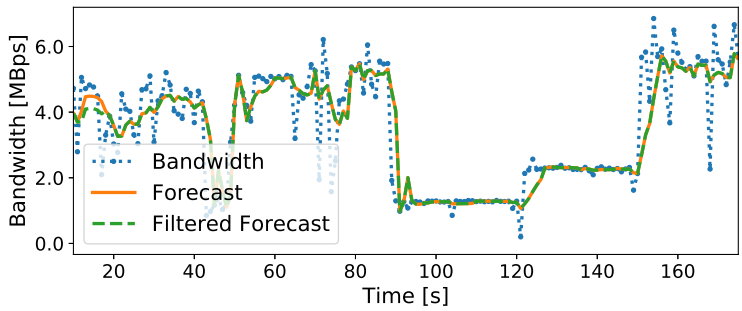


Figure 58: The controllers cannot use the control path (green) to fetch statistics as P4 only allows programming functionality for packets coming in a data-plane port. To this end, the controller requests statistics through a data-plane port (red) [Har+19a].

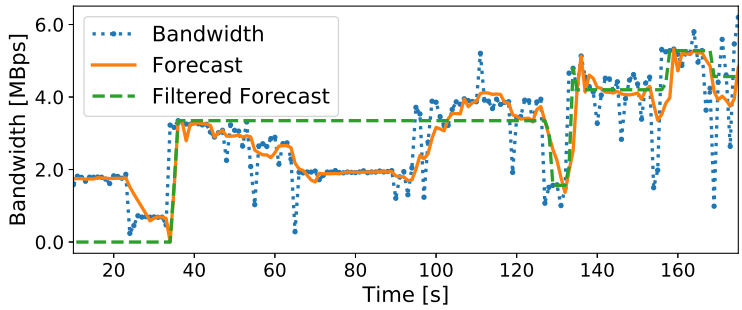
controller stored the parameters for the linear regression that were learned beforehand in the switch's registers, the P4 program on the switch executes an action containing the following steps whenever a statistic request reaches the switch.

1. Read the link byte counter and calculate the bandwidth for the current time interval.
2. Read and shift previous counters stored in registers in order to save the new counter.
3. Calculate the four previous bandwidths using the counter history.
4. Compute features based on the four previous and the current bandwidth.
5. Read the bias term, feature weight, and threshold registers.
6. Calculate the linear regression formula to estimate the improvement $\hat{i}$ using (Equation 27, p. 102).
7. Compare $\hat{i}$ with the threshold. If the estimated improvement exceeds the threshold, send the bandwidth measurement, otherwise the action is finished.

Following the described steps, the switch skips all measurements to the allowed extend defined by the improvement threshold. We show in Figure 59, how this threshold controls the frequency of measurement updates. Subfigure a compares the true bandwidth (dotted blue) with the forecast without filtering any measurement (solid orange) and the forecast when filtering measurement values with a predicted improvement below $25k$ (dashed green). The figure reveals that both forecasts overlap almost entirely and are close to the



(a) Improvement threshold of 25k.



(b) Improvement threshold of 500k

Figure 59: Comparison of the true bandwidth (dotted blue) and the forecast with (solid orange) and without filtering (dashed green). Small improvement thresholds keep the forecast close to an unfiltered forecast, whereas high thresholds, which allow a larger difference between the states in the network and the controller, lead to larger differences [Har+19a].

real bandwidth. Due to the relatively low threshold, which defines the difference between the state within the controller and the state seen by the switch, the error is tiny. In contrast to this, Subfigure b shows this comparison for a comparably high threshold of 500k. Here, we observe that the unfiltered forecast is again close to the true bandwidth in most cases; however,when filtering measurements with an insignificant improvement, the area between the forecast and the true bandwidth diverge more often. Nevertheless, on massive bandwidth changes, measurement updates trigger a forecast update. We refer to Appendix A.8 for the corresponding costs. It shows that low thresholds certainly produce much higher costs for statistic transmission and forecast update calculations than high thresholds – inversely proportional to the accuracy.

Challenges and Limitations

The described procedure filters measurements with a lowly estimated improvement on the programmable switch. Nevertheless, some challenges limit the plainness of the approach.

First of all, the set of operations supported by P4 is strongly limited to prevent users from adding complexity to the data-plane. For example, the support for multiplications is not fully given. P4's version 16 (P4v16) includes integer multiplications; yet, current P4 hardware switches multiply only with integers being the power of two, which is not applicable in our design. Older versions do not support multiplications at all. To this end, we use a workaround and map a multiplication to a sequence of shift-and-add operations[44] as known from CPUs [Bar02]. To estimate the improvement, we use this method to multiply feature weights with features. As this method can only multiply integers and feature weights are potentially decimals, we first multiply them with a large factor and cut the tail. Consequently, the estimated improvement is also shifted by the very same factor. We equalize this by shifting the threshold as well. Cutting the tail and shifting adds minor inaccuracy. In an auxiliary evaluation, we find when using a factor of 10, that the estimated improvement is almost equally good as using an accurate multiplication.

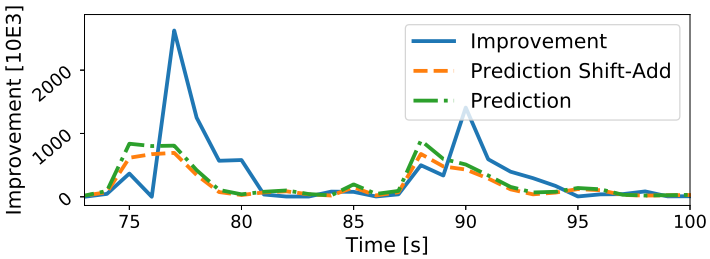


Figure 60: Comparison between the improvement prediction using exact multiplication (dash-dotted green) and using shift-and-add multiplication (dashed orange). The inaccuracy introduced when using shift-and-add-based multiplication is limited and allows the approach to be usable with any P4 switch [Har+19a].

Figure 60 shows a comparison between the shift-and-add based improvement prediction (dashed orange) and the accurate improvement prediction (dash-dotted green). The figure reveals that even for a low factor of 10, the estimations similar. For all other results, we use a factor of 1000, leading to even more accurate estimations. We find that P4's exemplary software switch (*behavior model*) has comparable performance using shift-and-add and normal multiplication: When using shift-and-add multiplications, we find a mean CPU utilization of 6,45% with confidence interval [6,26, 6,65] against 6,95% with [6,76, 7,14] for normal multiplications. Concluded, using shift-and-add instead of normal multiplications has no significant drawbacks and allows the approach to be usable with any P4 switches.

Despite that workaround, the lack of multiplications implies a lack of divisions. Therefore, whenever we talk about the bandwidth, we neglect the divi-

44 Shifting one bit left is a multiplication with two, shifting by 2 a multiplication with 2^2, etc.

sion by the time. We assume statistic requests to come with nearly equidistant time gaps. Thus, the division is not required when comparing historic bandwidths. However, we append timestamps to each statistic response such that the controller can calculate the true bandwidth.

Furthermore, it is worth noting that the filtering approach adds considerable memory consumption to the switch. For each counter pair (previous and current flow or link byte counter), we need four additional counters to store their history. In addition, for each of the four features, we need an additional register to store the feature weights. Plus two registers for the bias term and the threshold, we increase the memory consumption for two 64-bit counters to eleven 64-bit counters. The number of required counter and register strongly depend on the feature selection and has to be taken into account when using more complex features.

Lastly, we note that the approach is vulnerable to small steps in the measurements. If the new bandwidth slightly differs from the previous, it's improvement does not suffice to update the forecast. Multiple subsequent small increments in the bandwidth measurement can increase the difference between the state in the controller and the true bandwidths without the controller being aware of it. We leave solutions to this problem, e.g., with a fixed minimum update interval or by including the time from the last update in the features, for future work.

4.3.3 *Evaluation*

In this section, we evaluate the two presented filtering approaches. To show the applicability of filtering in general, we refer to the data center scenario in combination with *Configuration B* described in Section 3.3.1, p. 53. On top of this, we evaluate the extension of the filtering approach based on programmable switches within a small test arrangement. The controller is based on Ryu and uses P4Runtime[45] as control protocol. The forecast application calculates the *ARIMA*-based forecast using R's *forecast* library[46] and *rpy2*[47] as adapter between Python and R. As switch we use P4's software switch, denoted *behavior model*[48]. MININET [LHM10] connects the switch to two hosts that use *iperf* to send UDP traffic. Continuously, the sender sends flows with uniformly distributed length $\mathsf{U}_\mathsf{L}[0, 40]$ in seconds and uniformly distributed bandwidth $\mathsf{U}_\mathsf{B}[0, 40]$ in MBps. Thus, the traffic changes its intensity every 0...40 seconds to any bandwidth between 0...40 MBps.

First, we show the impact of filtering on the costs, majorly represented with the number of statistic requests that the switch transmits to the controller and the controller has to process. For comparison and to understand the

45 `https://p4.org/p4-runtime`, accessed 28 May 2019.
46 `https://cran.r-project.org/web/packages/forecast`, accessed 28 May 2019.
47 `https://rpy2.readthedocs.io/en/version_2.8.x`, accessed 28 May 2019.
48 `https://github.com/p4lang/behavioral-model`, accessed 28 May 2019.

trade-off, we show the accuracy represented as difference between the state observed without filtering and the state available in the controller with active filtering. In the case of using the application-aware approach, the accuracy is given as the difference between the actual bandwidth and the predicted bandwidth from the forecast application. Afterwards, we investigate the impact of the updates on the computational resources in terms of CPU utilization.

Accuracy and Cost Trade-Off when Filtering

The following figures show the costs and accuracy when measuring a randomly selected link within the federated data center networks every second ($f = 1/s$) for a runtime of 130 seconds. We denote the case that controllers measure the link without filtering with *Direct*. The parameter T within the figures is the measurement threshold, thus the minimal amount a measurement value must differ from the last measurement in order to be sent to the controller.

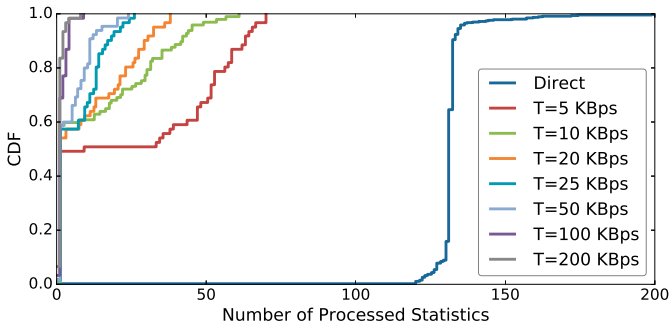
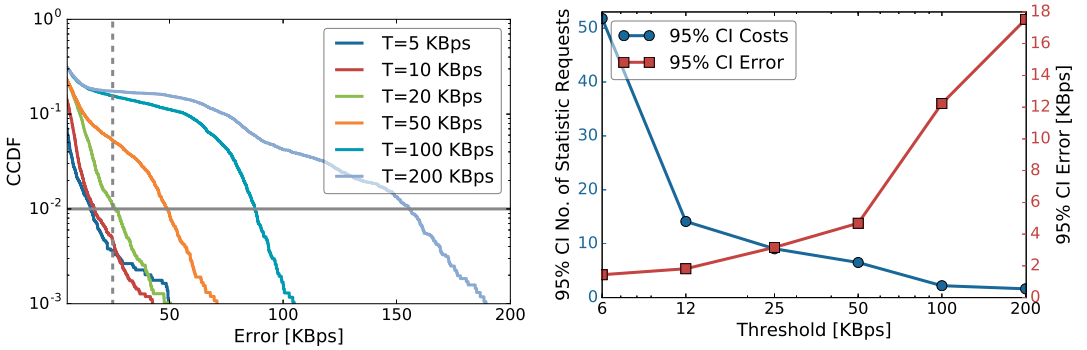


Figure 61: CDF of the number of processed statistics per controller with different delta thresholds (T). Using filtering reduces the number of statistics strongly. The larger the allowed threshold, the lower the costs [Har+19a].

Figure 61 shows the distribution of processed statistics per controller. First, considering the unfiltered case, the vast majority of values are 130 or close. As we measure every second for 130 seconds, this is the expected behavior. With active filtering, we see for all distributions, including the one with a minimal threshold of 5 Kbps that about 50% of all values are close to zero. Although this is a special case, it shows the particular strength of the filtering approach: The links do not seem to change its state for almost the whole runtime (e.g., as it does not see any traffic at all) such that there is no need to interrupt the controller with the measurements and force it to calculate a new forecast. On top of this, the number of processed statistics within the controllers depends on the used threshold. The larger the allowed difference between the state in the controller and the new measurement is, the fewer statistics sends the switch. By analogy to Figure 61, Appendix A.9 shows the number of processed statistic requests in a Box-Whisker-Plot, which high-

lights that the median is close to zero when filtering and how the maximum costs decrease in detail.

To correctly interpret the found results on the cost, we have to consider the accuracy as well. The frequency of measurement updates is directly inverse proportional to the costs as each update produces exactly one statistic transmission and its processing. In addition, we expect a connection between the update frequency and accuracy. Figure 62a depicts the distribution of the error between the filtered state in the controller and the state without filtering. Note the logarithmic y-scale of the complementary CDF. The figure reveals that, regardless of the threshold, ~30% come with negligible error. Subsequently, the threshold strongly influences the error distribution: The higher the allowed difference (threshold) the larger the error. Appendix A.9 contains the Box-Whishker-Plot for the same results, which can be considered to find the median more easily.



(a) CCDF for the error when filtering with different delta thresholds (T). The accuracy decreases with larger thresholds.

(b) Trade-off between the accuracy and costs.

Figure 62: Accuracy evaluation results as trade-off to the cost reduction [Har+19a].

Considering the vertical dashed gray line, which marks errors of 20 KBps, we observe that 99% of all values when using a threshold lower than 20 KBps have a much lower error. Directly comparing the costs with the accuracy, we find in Figure 62b the 95% confidence intervals (CI) for both when applying different thresholds. Note the logarithmic x-scale. The comparison shows that 95% of the costs can be expected to decrease strongly when increasing the threshold slightly in the beginning. In contrast to this, the 95% CI error remains rather stable for low thresholds and increases stronger for much larger thresholds. As a consequence, we claim that using low thresholds, which introduce only minor inaccuracies, already significantly reduce the costs and, therefore, should potentially be favored. However, the selection of the thresholds remains the operators choice as it strongly depends on the application.

Using the application-aware filtering approach based on programmable data-planes, the threshold has a different meaning. Instead of defining the

allowed difference between the last update and the newest measurement, it defines the minimal required *improvement* for a measurement to be considered important enough to be sent to the controller.

First, Figure 63 shows the cost reduction when filtering with respect to the application's quality. The figure shows the number of forecast updates that is equal to the number of statistic transmissions over a runtime of ~350 seconds. The unfiltered case (dotted blue) is linear as a new forecast is calculated using each measurement that comes every second regardless if they improve the application's quality or not. Furthermore, it is obvious that the smaller the threshold for a measurement's improvement is, the larger are the costs. When the threshold is small, measurements that improve the application's quality only little reach the controller. Logically, many more forecast recalculations are performed. The other way around, looking, for instance, in detail on a threshold of 200k (dotted purple), we see that the number of forecast updates reduces to roughly a third of the cost without filtering (at the end of the timeline slightly more than 100 updates are calculated instead of roundabout 350). With an improvement threshold of 800k, the costs are barely observable. To understand the cost reduction, we now consider the accuracy with respect to the improvement threshold.

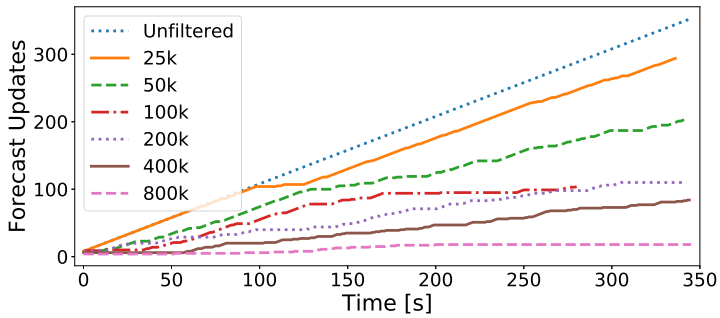


Figure 63: Number of conducted forecast updates over time when filtering with different improvement thresholds. The greater the required improvement, the lower the total costs [Har+19a].

Figure 64 shows the error between the bandwidth forecast and the true bandwidth on the investigated link. First of all, as expected, without filtering, we see the smallest error. However, the introduced error for thresholds under 200k is also negligible, if noticeable at all. Afterwards, the median error barely increases with a threshold of 400k. Further increasing the threshold up to 800k, we see an increase in the error. Comparing the cost reduction in Figure 63 with the achieved accuracy in Figure 64, it becomes clear that the costs decrease much fast than the accuracy suffers. For example, using a threshold of 100k barely degrades the accuracy, whereas the costs decrease by more than 50%. Leveraging machine learning to estimate the importance of a measurement to avoid irrelevant updates significantly increases the monitoring efficiency. The approach reduces costs on the networking resources

between the data- and control-plane as well as the controller's computational resources.

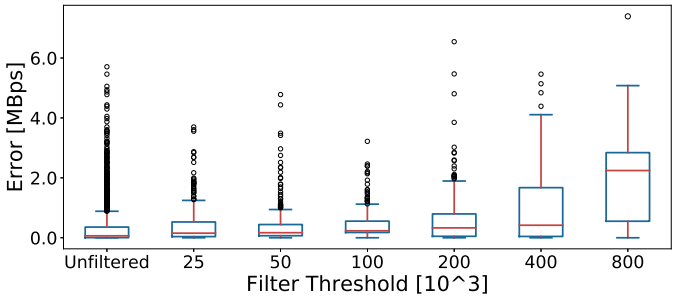
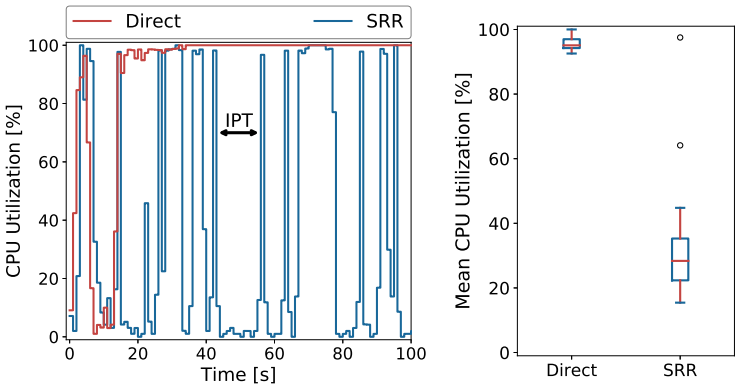


Figure 64: Costs when filtering with different improvement thresholds. For thresholds under 200, the accuracy remains stable, where previous figures show that the costs decrease strongly. With larger thresholds, the accuracy starts decreasing [Har+19a].

Influence on Computational Resources

So far, we showed the reduction of transmitted measurements and forecast updates. In addition, we now show the influence on the computational resources when calculating a CPU intense bandwidth forecast based on all given measurements. The following results are found using the filtering entity between the data- and control-plane that filters based on the similarity to previous measurements.

Subfigure a of Figure 65 shows exemplarily the CPU utilization over a runtime of 100 seconds. After an initial startup phase, the utilization is almost always at 100% when not filtering measurements (red, *Direct*). When filtering, we see that the utilization has peaks now and then. The peaks overlap with measurement updates that always trigger a forecast update on the controller. The length of the inter-peak time (marked with IPT) depends on the threshold. Larger thresholds that produce fewer measurement updates lead to longer IPTs and vice versa. The shorter IPTs are, the higher is the mean CPU utilization. Figure 65b contains statistical results for multiple evaluation runs. We observe a CPU utilization reduction from above 90% in median to ~30% when filtering. For the figure, a threshold of 20 KBps is used. Both figures show that the number of measurement updates strongly impact the computational resources. Concluded, considering all measurements does not always increase an application's accuracy but unnecessarily burdens resources. The proposed filtering approaches increase the efficiency sharply by dropping measurements with only few information content.



(a) Exemplary timeline for the CPU utilization.

(b) Distribution of the CPU utilization.

Figure 65: CPU utilization when calculating a forecast once a new measurement reaches the controller. Subfigure a shows an exemplary timeline: After a startup phase, the utilization when not filtering is close to 100%. With filtering, on every update there is a peak while the utilization is low between peaks. Subfigure b shows the distribution over several runs. It can be observed that the median decreases from over 90% to roughly 30% when filtering based on the similarity to previous measurements [Har+19a].

4.4 Summary

This chapter handled the second contribution of this thesis. It tackled the monitoring data collection process, particularly how this process can benefit from the softwarization of communication networks. Within the collection process, we identified three main aspects that were treated in this chapter: *(i) How* the controller should collect information; *(ii) Where* information should be captured; *(iii) Which* information is of relevance for the controllers. For the first aspect, we compared techniques, including traditional techniques, existing techniques specifically designed for SDNs, and potential future techniques. We found that each technique performs differently depending on the conditions and measurement requirements such that the concept of transitions to change between the active technique should be favored instead of trying to find one optimal technique for all situations. We exemplarily showed how the costs could be decreased while maintaining the accuracy in different situations by changing the applied technique. Concerning the second aspect, we followed the notion that not all measurement points provide the same quality of measurements. Therefore, we proposed to maximize the amount of information of potential measurement points by considering the topology when no knowledge on traffic in the network is available. Leveraging centrality metrics to estimate the importance of a node, we showed that a low number of measurement points selected according to our strategies, provide the same accuracy as high numbers of measurement points, which

measure everywhere. In addition, we showed that the placement based on centrality scores is equally good as placement strategies that have knowledge on traffic. For the third aspect, we followed the same notion as for the latter aspect: Not all measurements provide the same amount of information. We proposed different filtering mechanisms to prevent wasting networking and computational resources for statistics with low information gain. In a first step, an entity between the data- and control-plane filters statistics, which are not significantly different from previously transmitted statistics. Subsequently, we use a simple machine learning mechanism, a linear regression, to estimate the importance of a measurement for an application and leverage programmable switches to filter based on this estimation. We showed that both approaches allow decreasing the costs significantly while trading little accuracy. On top of that, using the application-aware filtering approach, we were able to maintain an application's accuracy while halving the costs.

5

Summary

In this section, we recap the goals of this thesis, our contributions concerning the goals and thereby answering the research questions. In addition, we sketch reasonable continuations and open questions in an outlook.

5.1 Contributions revisited

In Chapter 1, we identified a potential efficiency gap when monitoring the state in federations of Software-Defined Networks. SDNs provide a set of new, useful techniques to capture the network state and introduce a completely different architecture such that most existing mechanisms from legacy networks become inefficient, although they remain applicable. Existing monitoring approaches designed for SDNs do not consider the distributed architecture of networks (e.g., distributed data centers) and the physical distribution of the control-plane. However, this distribution opens up the potential for collaboration in monitoring of shared resources. Accordingly, the first goal of this thesis was to develop monitoring mechanisms that leverage the network's distributed nature, a task we tackled in Chapter 3 to answer the first research question. We followed the notion that multiple controllers of federated networks, which are willing to collaborate, possibly measure shared resources redundantly and, therefore, produce unnecessary overhead. To overcome this, we proposed approaches to coordinate monitoring applications of connected controllers and share statistical information that is of interest for multiple controllers. As the first step, we developed an intermediate monitoring entity between the data-plane and control-plane to perform monitoring tasks on behalf of the controllers. This centralized entity correlates registered tasks and aggregates them whenever possible. For example, this concerns tasks that measure the same resource, such as a flow traversing multiple adjacent networks, with overlapping task specifications, i.e., the same measurement frequency and measurement location. As a next step, we eliminated the entity, which constitutes a single point of failure that is vulnerable to faults and is not scalable. Therefore, we advanced the collaboration mechanism by pushing the aggregation functionality out of an additional intermediate entity onto the controllers. The resulting distributed system uses one of the participating controllers as coordinator. This collects monitoring interests, distributes monitoring responsibilities, and instructs controllers to share their measurements with interested controllers. Through the coordination of monitoring tasks, the system eliminates redundant measurements of the same

information. Using fairness enforcing schemes, the system distributes tasks in a way that the load on controllers and switches is balanced according to the chosen fairness metric. We further stabilized the system by developing a next variant that eliminates the central role of the coordinator. This design is robust against controller failures and network separation. To achieve this, we proposed a decentralized algorithm to coordinate controllers and assign monitoring task responsibilities. The algorithm also considers the controller load to avoid discrimination of single controllers.

In Chapter 4, we took a closer look at the statistics collection process conducted between controllers and the data-plane. We formulated three goals to answer the identified question on "*how* and *where* information must be *collected and selected*" within this process. First, in Section 4.1, we tackled the aspect of *how* information can be captured from the data-plane elements. Numerous techniques to measure statistics conceivable in future Softwarized Networks exist. They range from techniques originating from legacy networks to techniques particularly introduced by SDNs to techniques recently proposed by the research community. We selected four exemplary techniques from different categories to measure the loss on a link. Subsequently, we proved our notion that all of these techniques perform very differently depending on conditions and requirements. By investigating their performance and costs in different conditions, we concluded that none of the techniques is superior in every situation and a flexible exchange of the applied technique is preferable. Based on our findings, we gave a guideline on when to apply each technique to maintain the accuracy, while keeping the costs as low as possible. Second, the aspect of *where* information should be captured was handled in Section 4.2. Intending to increase the efficiency with a sensible measurement point selection, we targeted to maximize the information of a single measurement. Eventually, a small number of measurement points suffices to obtain the full state of the network with decent accuracy. To do so, we considered two scenarios: Having and not having knowledge on traffic in the network. Assuming having knowledge, we placed the measurement points with respect to the number of flows they cover in total. In contrast, without knowledge, we borrowed centrality scores from the social network theory to estimate the importance of a node in the network and assumed that nodes with central importance for the network provide the most information. Last, in Section 4.3, we tackled the aspect of *which* information to capture. Based on the same goal as before, we intended to gather as much information as possible, while reducing the required number of measurements. To this end, we developed mechanisms to filter measurements that contain only a small amount of information, thus saving networking and processing costs. In the first design, an intermediate entity between the data- and control-plane inspects measured values to decide whether it is worth being processed on the controller, e.g., depending on the difference to previous measurements. In the second step, we eliminated the new entity and pushed the filtering function-

ality to the data-plane such that switches filter statistics as soon as they are requested, to save transmission costs further. Moreover, the extended filtering does not only filter based on the measurement history but also according to the relevance for a particular application. Using linear regression, it estimates the improvement of the quality of an application for each measurement before dispatching it. Only if the switch finds a certain use for the application, it sends the measurement. To push this functionality to the data-plane, we created a novel approach leveraging programmable P4 switches.

5.2 Conclusions

In this work, we developed mechanisms for collaborative monitoring and to enhance the data collection process in federated SDNs, as described in the previous section. *Ultimately, the suggested approaches all significantly increase the efficiency of monitoring, thus increasing or maintaining the accuracy, while reducing the costs in terms of communication and processing overhead.* Considering the three developed evolutions of the collaboration methods, in our first contribution, we showed that the method successfully decreases the measurement costs. In our evaluation scenario, we were able to reduce the number of required statistic requests by half. Furthermore, additionally considering the introduced overhead for coordination and statistic sharing, we significantly decrease the total costs. On top of this, the developed method allows balancing the load on controllers and switches fairly. The fairness schemes particularly relieve controllers in focal positions, which see more flows than other controllers, from disproportional measurement load. Moreover, we showed that the accuracy in terms of staleness of measurements does not suffer for the vast majority of measurements.

Taking a closer look at the data collection process, we proposed methods, which sharply increase the efficiency. The flexible measurement technique selection achieves constant accuracy while always using the technique with the lowest costs. In addition, we successfully showed how to select the measurement points in order to receive the highest amount of information possible, with a small number of measurement points using fast placement strategies. Within various topologies, our evaluations showed that using the centrality-based placement achieves the same accuracy when using only a fraction of measurement points while needing only a fraction of measurement transmission costs. Regarding the selection of measurements, we could save costs by applying the proposed filtering mechanism. First, only forwarding measurement updates with significantly different values compared to previous measurements allows a significant reducing of the costs. Second, the filtering based on the application's improvement using programmable switches and leveraging machine learning shows – in the best configuration – a drop of the costs to half. Again, the accuracy was not affected at all.

In conclusion, all mechanisms successfully decrease the costs while respecting the accuracy. Both collaborative monitoring and the data collection procedure immensely benefit from the results in this thesis.

5.3 Outlook

In this work, we showed fundamental approaches to address research gaps to increase monitoring efficiency using collaboration and information-oriented data collection. These fundamentals provide a sound basis for ever-growing and highly dynamic networks, enabling them to master the rising monitoring load and complexity in the future.

The first contribution of this thesis motivates to investigate possibilities that increase the monitoring efficiency within cooperative Software-Defined Networks. In this work, we presented approaches to leverage coordination to eliminate redundant measurements within cooperative network parts. Further studies should include enabling collaboration for other aspects, e.g., using the information of adjacent networks to enrich incomplete information about the network. Recently, many works leverage machine learning to estimate a network state representation based on incomplete or uncertain data. We think that information from neighboring network parts can increase the accuracy of the state estimation.

Concerning the contributions in the context of monitoring data collection, we showed that the concept of transitions between techniques is a feasible approach to overcome challenges originating from highly dynamic demands and changing conditions. While we show the different behaviors of state estimation techniques and show how a transition-enabled monitoring system could increase the monitoring efficiency, open questions remain when developing such a system. The Collaborative Research Centre MAKI [Gro+13] is a viable starting point to tackle most of these challenges, including proactive transition planning, transitions on multiple coexisting and interacting layers, etc.

Lastly, in this work, we used programmable data-plane switches to improve the monitoring data collection process. As this field is still in its infancy, unexploited potentials to support monitoring must be addressed. This also holds true for the use of machine learning to support monitoring data collection and applications, as it develops into a seminal technology to tackle a broad spectrum of today's and future challenges of communication networks.

Bibliography

[AMM98] A. Adams, J. Mahdavi, and M. Mathis. “Creating a Scalable Architecture for Internet Measurement.” In: *IEEE Network* (1998), pp. 1–16.

[Afe+15] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff. “Sampling and Large Flow Detection in SDN.” In: *ACM SIGCOMM Computer Communication Review (CCR)* 45.4 (2015), pp. 345–346.

[Alt+19] B. Alt, M. Weckesser, C. Becker, M. Hollick, S. Kar, A. Klein, R. Klose, R. Kluge, H. Koeppl, B. Koldehofe, W. R. Khudabukhsh, M. Luthra, M. Mousavi, M. Mühlhäuser, M. Pfannemüller, A. Rizk, A. Schürr, and R. Steinmetz. “Transitions: A Protocol-Independent View of the Future Internet.” In: *Proceedings of the IEEE* 107.4 (2019), pp. 835–846.

[AB16] A. Atary and A. Bremler-Barr. “Efficient Round-Trip Time Monitoring in OpenFlow Networks.” In: *International Conference on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[Awe87] B. Awerbuch. “Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems.” In: *Symposium on Theory of Computing (STOC)*. ACM, 1987, pp. 230–240.

[BB13] F. Baccelli and P. Brémaud. *Elements of queueing theory: Palm Martingale calculus and stochastic recurrences*. Vol. 26. Springer Science & Business Media, 2013.

[Bac+07] F. Baccelli, S. Machiraju, D. Veitch, and J.C. Bolot. “On Optimal Probing for Delay and Loss Measurement.” In: *SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2007, pp. 291–302.

[BRA10] J.R. Ballard, I. Rae, and A. Akella. “Extensible and Scalable Network Monitoring Using OpenSAFE.” In: *Internet Network Management Conference on Research on Enterprise Networking (INM/WREN)*. USENIX Association, 2010, pp. 1–6.

[Ban+07] N. Bandi, A. Metwally, D. Agrawal, and A. El Abbadi. “Fast Data Stream Algorithms Using Associative Memories.” In: *International Conference on Management of Data (SIGMOD)*. ACM, 2007, pp. 247–256.

[BS04] P. Barford and J. Sommers. “Comparing Probe- and Router-based Packet-loss Measurement.” In: *IEEE Internet Computing* 8.5 (2004), pp. 50–56.

[Bar+13] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. “Dynamic Controller Provisioning in Software Defined Networks.” In: *Conference on Network and Service Management (CNSM)*. IFIP, 2013, pp. 18–25.

[Bar02] Z.F. Baruch. *Structure of computer systems*. UT Pres, 2002.

[BV02] P. Benko and A. Veres. “A Passive Method for Estimating End-to-End TCP Packet Loss.” In: *Global Telecommunications Conference (GLOBECOM)*. Vol. 3. IEEE, 2002, pp. 2609–2613.

[BAM10] T. Benson, A. Akella, and D.A. Maltz. “Network Traffic Characteristics of Data Centers in the Wild.” In: *SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2010, pp. 267–280.

[Ber+14a] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow, and G. Parulkar. “ONOS: Towards an Open, Distributed SDN OS.” In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 1–6.

[Ber+14b] M. Berman, J.S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. “GENI: A Federated Testbed for Innovative Network Experiments.” In: *Elsevir Computer Networks – Special issue on Future Internet Testbeds* 61 (2014), pp. 5–23.

[Bha+17] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. “Network Assisted Content Distribution for Adaptive Bitrate Video Streaming.” In: *Multimedia Systems Conference (MMSys)*. ACM, 2017, pp. 62–75.

[Bha46] A. Bhattacharyya. “On a Measure of Divergence between Two Multinomial Populations.” In: *ISI Sankhya: The Indian Journal of Statistics* 7.4 (1946), pp. 401–406.

[Bol93] J.C. Bolot. “End-to-end Packet Delay and Loss Behavior in the Internet.” In: *ACM SIGCOMM Computer Communication Review (CCR)* 23.4 (1993), pp. 289–298.

[Bor06] S.P. Borgatti. “Identifying Sets of Key Players in a Social Network.” In: *Springer Computational & Mathematical Organization Theory* 12.1 (2006), pp. 21–34.

[Bos+14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. “P4: Programming Protocol-independent Packet Processors.” In: *ACM SIGCOMM Computer Communication Review (CCR)* 44.3 (2014), pp. 87–95.

[BR13] Z. Bozakov and A. Rizk. “Taming SDN Controllers in Heterogeneous Hardware Environments.” In: *European Workshop on Software Defined Networks (EWSDN)*. IEEE, 2013, pp. 50–55.

[Boz+16] Z. Bozakov, A. Rizk, D. Bhat, and M. Zink. “Measurement-based Flow Characterization in Centrally Controlled Networks.” In: *International Conference on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[BM04] A. Broder and M. Mitzenmacher. “Network Applications of Bloom Filters: A Survey.” In: *Taylor & Francis: Internet Mathematics* 1.4 (2004), pp. 485–509.

[Bur+14] D. Burgstahler, N. Richerzhagen, F. Englert, R. Hans, and R. Steinmetz. “Switching Push and Pull: An Energy Efficient Notification Approach.” In: *Conference on Mobile Services (MS)*. IEEE, 2014, pp. 68–75.

[Can+06] G.R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. “Reformulating the Monitor Placement Problem: Optimal Network-wide Sampling.” In: *Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2006, pp. 1–12.

[CDJ17] F. Carpio, S. Dhahri, and A. Jukan. “VNF Placement with Replication for Load Balancing in NFV Networks.” In: *International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[Cas+07] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. "Ethane: Taking Control of the Enterprise." In: *ACM SIGCOMM Computer Communication Review (CCR)* 37.4 (2007), pp. 1–12.

[Cas+06] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, and S. Shenker. "SANE: A Protection Architecture for Enterprise Networks." In: *Security Symposium (Security)*. Vol. 49. USENIX Association, 2006, pp. 1–15.

[Cha+17] R. Challa, S. Jeon, D. S. Kim, and H. Choo. "CentFlow: Centrality-Based Flow Balancing and Traffic Distribution for Higher Network Utilization." In: *IEEE Access* 5 (2017), pp. 17045–17058.

[Cha+05] C. Chaudet, E. Fleury, I.G. Lassous, H. Rivano, and M.E. Voge. "Optimal Positioning of Active and Passive Monitoring Devices." In: *Conference on Emerging Network EXperiment and Technology (CoNEXT)*. ACM, 2005, pp. 71–82.

[CL00] M. Cheikhrouhou and J. Labetoulle. "An Efficient Polling Layer for SNMP." In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2000, pp. 477–490.

[Chi+15a] T. Chin, X. Mountrouidou, X. Li, and K. Xiong. "An SDN-supported collaborative approach for DDoS flooding detection and containment." In: *Military Communications Conference (MILCOM)*. IEEE, 2015, pp. 659–664.

[Chi+15b] T. Chin, X. Mountrouidou, X. Li, and K. Xiong. "Selective Packet Inspection to Detect DoS Flooding Using Software Defined Networking (SDN)." In: *Conference on Distributed Computing Systems Workshops (ICDCS Workshops)*. IEEE, 2015, pp. 95–99.

[Cho+14a] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park. "SuVMF: Software-Defined Unified Virtual Monitoring Function for SDN-based Large-scale Networks." In: *Conference on Future Internet Technologies (CFI)*. ACM, 2014, pp. 1–6.

[Cho+14b] T. Choi, S. Song, H. Park, S. Yoon, and S. Yang. "SUMA: Software-Defined Unified Monitoring Agent for SDN." In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–5.

[Cho+14c] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. "PayLess: A low cost network monitoring framework for Software Defined Networks." In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[Cis18] Cisco. "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021." In: *Cisco Global Cloud Index. White Paper.* (2018), pp. 1–46. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf.

[Cla04] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. 2004. URL: http://www.rfc-editor.org/rfc/rfc3954.txt.

[Col83] A. Colvin. "CSMA with Collision Avoidance." In: *Elsevir Computer Communications* 6.5 (1983), pp. 227–235.

[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.

[Cuz+12] A. Cuzzocrea, A. Papadimitriou, D. Katsaros, and Y. Manolopoulos. "Edge Betweenness Centrality: A Novel Algorithm for QoS-based Topology Control over Wireless Sensor Networks." In: *Elsevier Journal of Network and Computer Applications* 35.4 (2012), pp. 1210–1217.

[DR02] M. Dilman and D. Raz. "Efficient Reactive Monitoring." In: *IEEE Journal on Selected Areas in Communications (J-SAC)* 20.4 (2002), pp. 668–676.

[Dix+14] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella. "ElastiCon: An Elastic Distributed SDN Controller." In: *Symposium on Architectures for Networking and Communications Systems*. ACM/IEEE, 2014, pp. 17–27.

[Dol+09] S. Dolev, Y. Elovici, R. Puzis, and P. Zilberman. "Incremental Deployment of Network Monitors Based on Group Betweenness Centrality." In: *Elsevier Information Processing Letters* 109.20 (2009), pp. 1172–1176.

[DLT05] N. Duffield, C. Lund, and M. Thorup. "Estimating Flow Distributions from Sampled Flow Statistics." In: *IEEE/ACM Transactions on Networking (ToN)* 13.5 (2005), pp. 933–946.

[EB99] M.G. Everett and S.P. Borgatti. "The Centrality of Groups and Classes." In: *The Journal of Mathematical Sociology* 23.3 (1999), pp. 181–201.

[Fre78] L.C. Freeman. "Centrality in Social Networks Conceptual Clarification." In: *Elsevier Social Networks* 1.3 (1978), pp. 215–239.

[Fri+09] A. Friedl, S. Ubik, A. Kapravelos, M. Polychronakis, and E.P. Markatos. "Realistic Passive Packet Loss Measurement for High-Speed Networks." In: *Traffic Monitoring and Analysis*. Ed. by Maria Papadopouli, Philippe Owezarski, and Aiko Pras. Springer Berlin Heidelberg, 2009, pp. 1–7.

[Gao01] L. Gao. "On Inferring Autonomous System Relationships in the Internet." In: *IEEE/ACM Transactions on Networking (ToN)* 9.6 (2001), pp. 733–745.

[Ger17] A. Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017.

[Gio+14] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments." In: *Elsevier Computer Networks* 62 (2014), pp. 122–136.

[Gon+15] Y. Gong, W. Huang, W. Wang, and Y. Lei. "A survey on software defined networking and its applications." In: *Springer Frontiers of Computer Science* 9.6 (2015), pp. 827–845.

[Gre15] A. Greenberg. "SDN for the Cloud." In: *Keynote at the Conference on Special Interest Group on Data Communication (SIGCOMM)*. 2015. URL: https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/keynote.pdf.

[Gre+05] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. "A Clean Slate 4D Approach to Network Control and Management." In: *ACM SIGCOMM Computer Communication Review (CCR)* 35.5 (2005), pp. 41–54.

[Gro+13] C. Groß, D. Stingl, W. Effelsberg, and R. Steinmetz. "Neuer DFG-Sonderforschungsbereich an der Technischen Universität Darmstadt." In: *PIK-Praxis der Informationsverarbeitung und Kommunikation* 36.1 (2013), pp. 65–66.

[Han+15] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. "Network Function Virtualization: Challenges and Opportunities for Innovations." In: *IEEE Communications Magazine* 53.2 (2015), pp. 90–97.

[HTK13] R. Hand, M. Ton, and E. Keller. "Active Security." In: *SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. ACM, 2013, pp. 1–7.

[Har17] R. Hark. *GitHub: In-band SDN Control Communication*. 2017. URL: https://github.com/R./In-band-SDN-Control-Communication.

[Har+17a] R. Hark, Rizk A, N. Richerzhagen, B. Richerzhagen, and R. Steinmetz. "Isolated In-Band Communication for Distributed SDN Controllers." In: *IFIP Networking 2017 Conference and Workshops: IFIP/TC6 Networking Poster and Demo Session*. IEEE, 2017, pp. 1–2.

[Har+18a] R. Hark, N. Aerts, D. Hock, N. Richerzhagen, A. Rizk, and R. Steinmetz. "Reducing the Monitoring Footprint on Controllers in Software-Defined Networks." In: *IEEE Transactions on Network and Service Management (TNSM), Special Issue on Novel Techniques for Managing Softwarized Networks* 15.4 (2018), pp. 1264–1276.

[Har+19a] R. Hark, D. Bhat, M. Zink, R. Steinmetz, and A. Rizk. *Preprocessing Monitoring Information on the SDN Data-Plane using P4*. Currently under review. 2019.

[Har+18b] R. Hark, M. Ghanmi, S. Kar, N. Richerzhagen, A. Rizk, and R. Steinmetz. "Representative Measurement Point Selection to Monitor Software-Defined Networks." In: *Conference on Local Computer Networks (LCN)*. IEEE, 2018, pp. 511–518.

[Har+17b] R. Hark, N. Richerzhagen, B. Richerzhagen, A. Rizk, and R. Steinmetz. "Towards an Adaptive Selection of Loss Estimation Techniques in Software-Defined Networks." In: *IFIP Networking 2017 Conference (NETWORKING)*. IEEE, 2017, pp. 1–9.

[Har+16] R. Hark, D. Stingl, N. Richerzhagen, K. Nahrstedt, and R. Steinmetz. "DistTM: Collaborative Traffic Matrix Estimation in Distributed SDN Control Planes." In: *IFIP Networking 2016 Conference (NETWORKING)*. IEEE, 2016, pp. 82–90.

[Har+19b] R. Hark, K. Tounsi, A. Rizk, and R. Steinmetz. "Decentralized Collaborative Flow Monitoring in Distributed SDN Control-Planes." In: *GI/ITG Conference on Networked Systems (NetSys)*. IEEE, 2019, pp. 1–8.

[HH10] D. Harris and S. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, 2010, p. 129.

[HG12] S. Hassas Yeganeh and Y. Ganjali. "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2012, pp. 19–24.

[HH08] G. Hasslinger and O. Hohlfeld. "The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet." In: *Conference on Measurement, Modelling and Evalutation of Computer and Communication Systems (MMB)*. VDE GI/ITG, 2008, pp. 1–15.

[HSM12] B. Heller, R. Sherwood, and N. McKeown. "The Controller Placement Problem." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2012, pp. 7–12.

[HV06] N. Hohn and D. Veitch. "Inverting Sampled Traffic." In: *IEEE/ACM Transactions on Networking (ToN)* 14.1 (2006), pp. 68–80.

[Hor+82] S. Horing, J. Z. Menard, R. E. Staehler, and B. J. Yokelson. “Stored Program Controlled Network: Overview.” In: *The Bell System Technical Journal* 61.7 (1982), pp. 1579–1588.

[HLB18] Qun Huang, Patrick P. C. Lee, and Yungang Bao. “Sketchlearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference.” In: *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018, pp. 576–590.

[Jai90] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1990.

[Jai+13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. “B4: Experience with a Globally-deployed Software Defined Wan.” In: *ACM SIGCOMM Computer Communication Review (CCR)* 43.4 (2013), pp. 3–14.

[JYR11] L. Jose, M. Yu, and J. Rexford. “Online Measurement of Large Traffic Aggregates on Commodity Switches.” In: *Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*. USENIX Association, 2011, pp. 1–6.

[Joy+87] J. Joyce, G. Lomow, K. Slind, and B. Unger. “Monitoring Distributed Systems.” In: *ACM Transactions Computer Systems (TOCS)* 5.2 (1987), pp. 121–150.

[Kat+16] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. “HULA: Scalable Load Balancing Using Programmable Data Planes.” In: *SIGCOMM Symposium on SDN Research (SOSR)*. 10. ACM, 2016, pp. 1–12.

[Kim+04] M.S. Kim, H.J. Kong, S.C. Hong, S.H. Chung, and J.W. Hong. “A Flow-based Method for Abnormal Network Traffic Detection.” In: *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2004, pp. 599–612.

[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 (J-SAC)* 29.9 (2011), pp. 1765–1775.

[Kop+10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. “Onix: A Distributed Control Platform for Large-scale Production Networks.” In: *Conference on Operating Systems Design and Implementation*. USENIX Association, 2010, pp. 351–364.

[Koz+16] U. C. Kozat, G. Liang, K. Kokten, and J. Tapolcai. “On Optimal Topology Verification and Failure Localization for Software Defined Networks.” In: *IEEE/ACM Transactions on Networking (ToN)* 24.5 (2016), pp. 2899–2912.

[KCG14] A. Krishnamurthy, S.P. Chandrabose, and A. Gember-Jacobson. “Pratyaastha: An Efficient Elastic Distributed SDN Control Plane.” In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 133–138.

[Kum+04] A. Kumar, M. Sung, J. Xu, and J. Wang. “Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution.” In: *ACM SIGMETRICS Performance Evaluation Review* 32.1 (2004), pp. 177–188.

[Kun+18] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz. "P4-CoDel: Active Queue Management in Programmable Data Planes." In: *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–4.

[KR12] J. Kurose and K. Ross. *Computer Networks: A Top-Down Approach Featuring the Internet*. 6th ed. Addison-Wesley, 2012.

[Lam01] L. Lamport. "Paxos Made Simple." In: *ACM Sigact News* 32.4 (2001), pp. 18–25.

[Lan+15] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks." In: *IEEE Transactions on Network and Service Management (TNSM)* 12.1 (2015), pp. 4–17.

[LHM10] B. Lantz, B. Heller, and N. McKeown. "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks." In: *SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. ACM, 2010, pp. 1–6.

[Lev+12] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2012, pp. 1–6.

[Li+16] Y. Li, R. Miao, C. Kim, and M. Yu. "LossRadar: Fast Detection of Lost Packets in Data Center Networks." In: *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2016, pp. 481–495.

[Lim12] T.A. Limoncelli. "OpenFlow: A Radical New Idea in Networking." In: *Communications of the ACM* 55.8 (2012), pp. 42–47.

[LC06] Y.J. Lin and M.C. Chan. "A Scalable Monitoring Approach Based on Aggregation and Refinement." In: *IEEE Journal on Selected Areas in Communications* 20.4 (2006), pp. 677–690.

[Liu+16] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon." In: *Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016, pp. 101–114.

[Liu+15] Z. Liu, G. Vorsanger, V. Braverman, and V. Sekar. "Enabling a "RISC" Approach for Software-Defined Monitoring Using Universal Streaming." In: *SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. ACM, 2015, pp. 1–7.

[LLC15] SDxCentral LLC. *Market Report: SDN Controllers Report*. Tech. rep. 2015.

[Loc11] T. Locher. "Finding Heavy Distinct Hitters in Data Streams." In: *Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2011, pp. 299–308.

[Mac+07] S. Machiraju, D. Veitch, F. Baccelli, and J.C. Bolot. "Adding Definition to Active Probing." In: *ACM SIGCOMM Computer Communication Review (CCR)* 37.2 (2007), pp. 17–28.

[MS93] M. Mansouri-Samani and M. Sloman. "Monitoring Distributed Systems." In: *IEEE Network* 7.6 (1993), pp. 20–30.

[MS15] M. Markovitch and S. Schmid. "SHEAR: A Highly Available and Flexible Network Architecture Marrying Distributed and Logically Centralized Control Planes." In: *Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 78–89.

[McK+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks." In: *ACM SIGCOMM Computer Communication Review (CCR)* 38.2 (2008), pp. 69–74.

[Med+02] A. Medina, C. Fraleigh, N. Taft, S. Bhattacharyya, and C. Diot. "Taxonomy of IP Traffic Matrices." In: *Proceedings of the SPIE: Scalability and Traffic Control in IP Networks II*. Vol. 4868. 2002, pp. 200–214.

[Meg+16] P. Megyesi, A. Botta, G. Aceto, A. Pescape, and S. Molnar. "Available Bandwidth Measurement in Software Defined Networks." In: *Symposium on Applied Computing (SAC)*. ACM, 2016, pp. 651–657.

[Mos+14] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. "DREAM: Dynamic Resource Allocation for Software-Defined Measurement." In: *ACM SIGCOMM Computer Communication Review (CCR)* 44.4 (2014), pp. 419–430.

[Mos+15] M. Moshref, M. Yu, R. Govindan, and A. Vahdat. "SCREAM: Sketch Resource Allocation for Software-Defined Measurement." In: *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2015, pp. 1–13.

[NKS00] B.S. Nataraj, S. Khanna, and V. Srinivasan. *Ternary content addressable memory cell*. US Patent 6,154,384. 2000.

[NFV12] ETSI Industry Specification Group NFV. "Network Functions Virtualisation – Introductory White Paper." In: *ESTI White Paper* 1 (2012). URL: http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[OO14] D. Ongaro and J. Ousterhout. "In Search of an Understandable Consensus Algorithm." In: *Annual Technical Conference (ATC)*. USENIX Association, 2014, pp. 305–319.

[Ope12] Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Tech. rep. 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[Pap+06] A. Papadogiannakis, A. Kapravelos, M. Polychronakis, E. P. Markatos, and A. Ciuffoletti. "Passive End-to-End Packet Loss Estimation for Grid Traffic Monitoring." In: *CoreGRID Integration Workshop*. 2006, pp. 1–12.

[PCD03] K. Papagiannaki, R. Cruz, and C. Diot. "Network Performance Monitoring at Small Time Scales." In: *SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2003, pp. 295–300.

[Pax97] V. Paxson. "End-to-end Internet Packet Dynamics." In: *ACM SIGCOMM Computer Communication Review (CCR)* 27.4 (1997), pp. 139–152.

[Pfa+12] B. Pfaff et al. *OpenFlow Switch Specification, Version 1.3.0*. Tech. rep. 2012.

[Pfa+15a] B. Pfaff et al. *OpenFlow Switch Specification, Version 1.5.1*. Tech. rep. 2015.

[Pfa+15b] B. Pfaff, J. Pettit, T. Koponen, E.J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. "The Design and Implementation of Open vSwitch." In: *Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2015, pp. 117–130.

[PL04] P. Phaal and M. Lavine. *sFlow Version 5*. Tech. rep. 2004. URL: https://sflow.org/sflow_version_5.txt.

[PPM01] P. Phaal, S. Panchen, and N. McKee. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC 3176. 2001. URL: http://www.rfc-editor.org/rfc/rfc3176.txt.

[PTK13] X. T. Phan, N. Thoai, and P. Kuonen. “A Collaborative Model for Routing in Multi-domains OpenFlow Networks.” In: *Conference on Computing, Management and Telecommunications (ComManTel)*. IEEE, 2013, pp. 278–283.

[PBL14] K. Phemius, M. Bouet, and J. Leguay. “DISCO: Distributed Multi-domain SDN Controllers.” In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[PAM17] D.A. Popescu, G. Antichi, and A.W. Moore. “Enabling Fast Hierarchical Heavy Hitter Detection Using Programmable Data Planes.” In: *SIGCOMM Symposium on SDN Research (SOSR)*. ACM, 2017, pp. 191–192.

[PS06] A.G. Prieto and R. Stadler. “Adaptive Distributed Monitoring with Accuracy Objectives.” In: *SIGCOMM Workshop on Internet Network Management (INM)*. ACM, 2006, pp. 65–70.

[R00] Britta R. “Eigenvector-Centrality – A Node-Centrality?” In: *Springer Social Networks* 22.4 (2000), pp. 357–365.

[Ras+14] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. “Planck: Millisecond-scale Monitoring and Control for Commodity Networks.” In: *Conference of the Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2014, pp. 407–418.

[RSC14] D. Raumer, L. Schwaighofer, and G. Carle. “MonSamp: A distributed SDN application for QoS monitoring.” In: *Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2014, pp. 961–968.

[RLH06] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. 2006. URL: http://www.rfc-editor.org/rfc/rfc4271.txt.

[Rib+06] B. Ribeiro, D. Towsley, T. Ye, and J.C. Bolot. “Fisher Information of Sampled Packets: An Application to Flow Size Estimation.” In: *SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2006, pp. 15–26.

[Ric+16a] B. Richerzhagen, N. Richerzhagen, S. Schonherr, R. Hark, and R. Steinmetz. “Stateless Gateways - Reducing Cellular Traffic for Event Distribution in Mobile Social Applications.” In: *Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–9.

[Ric+16b] B. Richerzhagen, N. Richerzhagen, J. Zobel, S. Schönherr, B. Koldehofe, and R. Steinmetz. “Seamless Transitions between Filter Schemes for Location-Based Mobile Applications.” In: *Conference on Local Computer Networks (LCN)*. IEEE, 2016, pp. 348–356.

[Ric+15a] N. Richerzhagen, D. Stingl, B. Richerzhagen, A. Mauthe, and R. Steinmetz. “Adaptive Monitoring for Mobile Networks in Challenging Environments.” In: *Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–8.

[Ric+15b] N. Richerzhagen, D. Stingl, B. Richerzhagen, A. Mauthe, and R. Steinmetz. “Adaptive Monitoring for Mobile Networks in Challenging Environments.” In: *Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–8.

[RR16] F.J. Ros and P.M. Ruiz. “On Reliable Controller Placements in Software-Defined Networks.” In: *Elsevir Computer Communications* 77 (2016), pp. 41–51.

[Ros+96] S.M. Ross, J.J. Kelly, R.J. Sullivan, W.J. Perry, D. Mercer, R.M. Davis, T.D. Washburn, E.V. Sager, J.B. Boyce, and V.L. Bristow. *Stochastic processes*. Vol. 2. Wiley New York, 1996.

[Roy+15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A.C. Snoeren. "Inside the Social Network's (Datacenter) Network." In: *ACM SIGCOMM Computer Communication Review (CCR)* 45.4 (2015), pp. 123–137.

[Sab77] J.L.M. Saboia. "Autoregressive Integrated Moving Average (ARIMA) Models for Birth Forecasting." In: *Taylor & Francis Journal of the American Statistical Association* 72.358 (1977), pp. 264–270.

[Sav99] S. Savage. "Sting: A TCP-based Network Measurement Tool." In: *Symposium on Internet Technologies and Systems (USITS)*. USENIX Association, 1999, pp. 1–9.

[SSC15] L. Schiff, S. Schmid, and M. Canini. "Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network." In: *SIGCOMM Symposium on SDN Research (SOSR)*. ACM, 2015, pp. 1–2.

[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.

[SOS13] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. "SDN Security: A Survey." In: *SDN for Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–7.

[SRZ10] V. Sekar, M. K. Reiter, and H. Zhang. "Revisiting the Case for a Minimalist Approach for Network Flow Monitoring." In: *Conference on Internet Measurement (IMC)*. ACM, 2010, pp. 328–341.

[Sek+08] V. Sekar, M.K. Reiter, W. Willinger, H. Zhang, R.R. Kompella, and D.G. Andersen. "CSAMP: A System for Network-wide Flow Monitoring." In: *Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2008, pp. 233–246.

[AlS+14] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. "OpenVirteX: Make Your Virtual SDNs Programmable." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 25–30.

[SA12] S.H. Shen and A. Akella. "DECOR: A Distributed Coordinated Resource Monitoring System." In: *International Workshop on Quality of Service (IWQoS)*. IEEE, 2012, pp. 1–9.

[She+09] R. Sherwood, G. Gibb, K.K. Yap, M. Casado, N. Mckeown, and G. Parulkar. *FlowVisor: A Network Virtualization Layer*. Tech. rep. 2009, pp. 1–15.

[SG13] S. Shirali-Shahreza and Y. Ganjali. "FleXam: Flexible Sampling Extension for Monitoring and Security Applications in Openflow." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2013, pp. 167–168.

[Sie+14a] M. Siebenhaar, U. Lampe, D. Schuller, and R. Steinmetz. "R.ust Cloud Monitor Placement for Availability Verification." In: *Conference on Cloud Computing and Services Science (CLOSER)*. SCITEPRESS - Science and Technology Publications, 2014, pp. 193–198.

[Sie+14b] M. Siebenhaar, D. Schuller, O. Wenge, and R. Steinmetz. "Heuristic Approaches for R.ust Cloud Monitor Placement." In: *Service-Oriented Computing*. Ed. by X. Franch, Aditya K. Ghose, Grace A. Lewis, and S. Bhiri. Springer, 2014, pp. 321–335.

[Siv+16] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. "Programmable Packet Scheduling at Line Rate." In: *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016, pp. 44–57.

[Siv+17] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. "Heavy-Hitter Detection Entirely in the Data Plane." In: *SIGCOMM Symposium on SDN Research (SOSR)*. ACM, 2017, pp. 164–176.

[SBB13] R. Skowyra, S. Bahargam, and A. Bestavros. "Software-Defined IDS for Securing Embedded Mobile Devices." In: *High Performance Extreme Computing Conference (HPEC)*. IEEE, 2013, pp. 1–7.

[Sta13] W. Stallings. "Software-Defined Networks and OpenFlow." In: *Citeseer The Internet Protocol Journal* 16.1 (2013), pp. 2–14.

[Suh+06] K. Suh, Y. Guo, J. Kurose, and D. Towsley. "Locating network monitors: Complexity, heuristics, and coverage." In: *Elsevier Computer Communications* 29.10 (2006), pp. 1564–1577.

[TR13] A. TaheriMonfared and C. Rong. "Multi-tenant Network Monitoring Based on Software Defined Networking." In: *On the Move Federated Conferences (OTM)*. Ed. by R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. De Leenheer, and D. Dou. Springer, 2013, pp. 327–341.

[TC11] A.S. Tam and H.J. Chao. "Use of Devolved Controllers in Data Center Networks." In: *Conference on Computer Communications Workshops (INFOCOM Workshops)*. IEEE, 2011, pp. 596–601.

[Tan+16] G. Tangari, M. Charalambides, D. Tuncer, and G. Pavlou. "Decentralized Solutions for Monitoring Large-Scale Software-Defined Networks." In: *Conference on Autonomous Infrastructure, Management, and Security: Management and Security in the Age of Hyperconnectivity (AIMS)*. Ed. by R. Badonnel, R. Koch, A. Pras, M. Drasar, and B. Stiller. Springer, 2016, pp. 47–51.

[Tan+17a] G. Tangari, M. Charalambides, D. Tuncer, and G. Pavlou. "Adaptive Traffic Monitoring for Software Dataplanes." In: *Conference on Network and Service Management (CNSM)*. IEEE, 2017, pp. 1–9.

[Tan+17b] G. Tangari, D. Tuncer, M. Charalambides, and G. Pavlou. "Decentralized Monitoring for Large-Scale Software-Defined Networks." In: *Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 289–297.

[Tan+14] E. Tantar, M.R. Palattella, T. Avanesov, M. Kantor, and T. Engel. "Cognition: A Tool for Reinforcing Security in Software Defined Networks." In: *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*. Ed. by A.A. Tantar, E. Tantar, J.Q. Sun, W. Zhang, Q. Ding, O. Schütze, M. Emmerich, P. Legrand, P. Del Moral, and C.A. Coello Coello. Springer, 2014, pp. 61–78.

[Ten+97] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. "A Survey of Active Network Research." In: *IEEE Communications Magazine* 35.1 (1997), pp. 80–86.

[Ter+99] A. Terzis, L. Wang, J. Ogawa, and L. Zhang. "A two-tier resource management model for the Internet." In: *Global Telecommunications Conference (GLOBECOM))*. IEEE, 1999, pp. 1779–1791.

[TG10] A. Tootoonchian and Y. Ganjali. “HyperFlow: A Distributed Control Plane for OpenFlow.” In: *Internet Network Management Conference on Research on Enterprise Networking (INM/WREN)*. USENIX Association, 2010, pp. 1–6.

[TGG10] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. “OpenTM: Traffic Matrix Estimator for OpenFlow Networks.” In: *Passive and Active Measurement (PAM)*. Ed. by A. Krishnamurthy and B. Plattner. Springer Berlin Heidelberg, 2010, pp. 201–210.

[Tun+15] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou. “Adaptive Resource Management and Control in Software Defined Networks.” In: *IEEE Transactions on Network and Service Management (TNSM)* 12.1 (2015), pp. 18–33.

[Tun+13] P. Tune, M. Roughan, H. Haddadi, and O. Bonaventure. “Internet traffic matrices: A primer.” In: *ACM SIGCOMM Recent Advances in Networking* 1 (2013), pp. 1–56.

[TV11] P. Tune and D. Veitch. “Fisher Information in Flow Size Distribution Estimation.” In: *IEEE Transactions on Information Theory* 57.10 (2011), pp. 7011–7035.

[vDK14] N.L.M. van Adrichem, C. Doerr, and F.A. Kuipers. “OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks.” In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–8.

[Wan+16] T. Wang, F. Liu, J. Guo, and H. Xu. “Dynamic SDN Controller Assignment in Data Center Networks: Stable Matching with Transfers.” In: *International Conference on Computer Communications (INFOCOM)*. IEEE, 2016, pp. 1–9.

[Wan+13] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang. “NetFuse: Short-Circuiting Traffic Surges in the Cloud.” In: *International Conference on Communications (ICC)*. IEEE, 2013, pp. 3514–3518.

[WHP99] B. Wijnen, D. Harrington, and R. Presuhn. *An Architecture for Describing SNMP Management Frameworks*. RFC 2571. 1999. URL: `http://www.rfc-editor.org/rfc/rfc2571.txt`.

[YKT99] M. Yajnik, J. Kurose, and D. Towsley. “Measurement and Nodelling of the Temporal Dependence in Packet Loss.” In: *International Conference on Computer Communications (INFOCOM)*. IEEE, 1999, pp. 345–352.

[Yan+18] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. “Elastic Sketch: Adaptive and Fast Network-wide Measurements.” In: *Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018, pp. 561–575.

[Yin+12] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi. *SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains*. Internet-Draft draft-yin-sdn-sdni-00. IETF Secretariat, 2012. URL: `https://tools.ietf.org/html/draft-yin-sdn-sdni-00`.

[Yoo+17] S. Yoon, T. Ha, S. Kim, and H. Lim. “Scalable Traffic Sampling Using Centrality Measure on Software-Defined Networks.” In: *IEEE Communications Magazine* 55.7 (2017), pp. 43–49.

[Yu+15] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H.V. Madhyastha. “Software-Defined Latency Monitoring in Data Center Networks.” In: *Passive and Active Measurement (PAM)*. Ed. by J. Mirkovic and Y. Liu. Springer, 2015, pp. 360–372.

[Yu+13] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H.V. Madhyastha. "FlowSense: Monitoring Network Utilization with Zero Measurement Cost." In: *Passive and Active Measurement (PAM)*. Ed. by M. Roughan and Rocky Chang. Springer Berlin Heidelberg, 2013, pp. 31–41.

[YJM13] M. Yu, L. Jose, and R. Miao. "Software Defined Traffic Measurement with OpenSketch." In: *Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2013, pp. 29–42.

[YQL14] Y. Yu, C. Qian, and X. Li. "Distributed and Collaborative Traffic Monitoring in Software Defined Networks." In: *SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 85–90.

[Zaa+14] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou. "OrchSec: An Orchestrator-based Architecture for Enhancing Network-Security using Network Monitoring and SDN Control Functions." In: *Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–9.

[Zha+16] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang. "Mind the Gap: Monitoring the Control-Data Plane Consistency in Software Defined Networks." In: *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2016, pp. 19–33.

[Zha13] Y. Zhang. "An Adaptive Flow Counting Method for Anomaly Detection in SDN." In: *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2013, pp. 25–30.

[ZD01] Y. Zhang and N. Duffield. "On the Constancy of Internet Path Properties." In: *SIGCOMM Workshop on Internet Measurement (IMW)*. ACM, 2001, pp. 197–211.

[ZH07] Y. Zhou and J. He. "A Runtime Analysis of Evolutionary Algorithms for Constrained Optimization Problems." In: *IEEE Transactions on Evolutionary Computation* 11.5 (2007), pp. 608–619.

Internet resources have been accessed by June 2019.

A

Appendix

A.1 OpenFlow Action Header Types

The actions given in Listing 2 can be called according to the OpenFlow switch specification v1.3.0.

Listing 2: Exemplary task registration message [Pfa+12].

```
enum ofp_action_type {
  OFPAT_OUTPUT        = 0,  /* Output to switch port. */
  OFPAT_COPY_TTL_OUT  = 11, /* Copy TTL "outwards" -- from next-to-outermost to
      outermost */
  OFPAT_COPY_TTL_IN   = 12, /* Copy TTL "inwards" -- from outermost to next-to-
      outermost */
  OFPAT_SET_MPLS_TTL  = 15, /* MPLS TTL */
  OFPAT_DEC_MPLS_TTL  = 16, /* Decrement MPLS TTL */
  OFPAT_PUSH_VLAN     = 17, /* Push a new VLAN tag */
  OFPAT_POP_VLAN      = 18, /* Pop the outer VLAN tag */
  OFPAT_PUSH_MPLS     = 19, /* Push a new MPLS tag */
  OFPAT_POP_MPLS      = 20, /* Pop the outer MPLS tag */
  OFPAT_SET_QUEUE     = 21, /* Set queue id when outputting to a port */
  OFPAT_GROUP         = 22, /* Apply group. */
  OFPAT_SET_NW_TTL    = 23, /* IP TTL. */
  OFPAT_DEC_NW_TTL    = 24, /* Decrement IP TTL. */
  OFPAT_SET_FIELD     = 25, /* Set a header field using OXM TLV format. */
  OFPAT_PUSH_PBB      = 26, /* Push a new PBB service tag (I-TAG) */
  OFPAT_POP_PBB       = 27, /* Pop the outer PBB service tag (I-TAG) */
  OFPAT_EXPERIMENTER  = 0xffff
};
```

A.2 Measurement Overhead per Mean Flow Duration.

Figure 66 shows the monitoring costs in terms of bytes for conducted measurements per controller on the y-axis when using the decentral algorithm to coordinate monitoring of multiple controllers (cf. Section 3.2.2). The x-axis lists different mean flow durations. We observe a significant reduction of the monitoring costs: First, we observe a general reduction of costs using the coordination approach (blue) compared to the uncoordinated case (hashed orange) for all flow durations. Second, by focusing on the median within the figure, we see that the relative cost reduction increases when the mean flow duration is longer. As longer flows produce more measurements when measured periodically, the total number of eliminated redundant measurements increases proportionally.

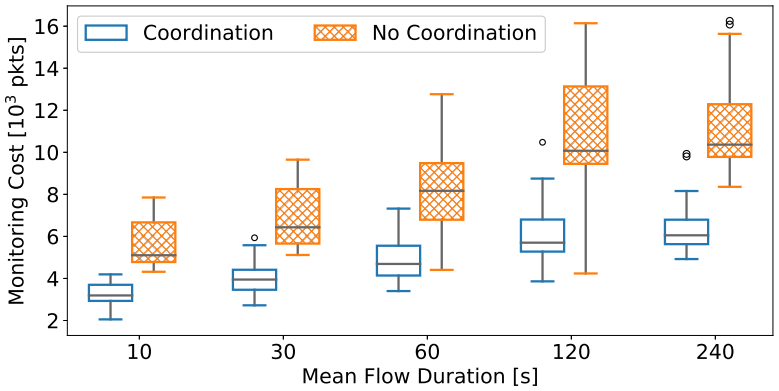


Figure 66: Comparison of the monitoring costs in terms of bytes for measurements between coordinated and uncoordinated runs using different mean flow durations [Har+16a].

A.3 Measurement Overhead per Coordinaton Threshold.

Figure 67 shows the monitoring overhead in bytes per controller on the y-axis when using DISTTM (cf. Section 3.2). The x-axis depicts different coordination thresholds and for each threshold, the figure shows four different bars representing different measurement periods (inverse update frequencies). The larger the measurement period (T), the less costs are produced in total independently of the thresholds. This is due to the decrementation of the total number of measurements if the frequency drops. Furthermore, comparing the first bar (without coordination) to the others, the figure shows that the total savings are smaller if the period is shorter since there are more measurements in total. Also, it reveals that the operating point, which is the coordination threshold that leads to the smallest total overhead, changes with different periods. With a period of $T = 500ms$, we have the lowest overhead with a threshold of one. In contrast, with a larger period of $T = 5000ms$, we see that the overhead reduces more with higher thresholds, before it increases again. Therefore, a threshold of three is preferable as operation point in this case.

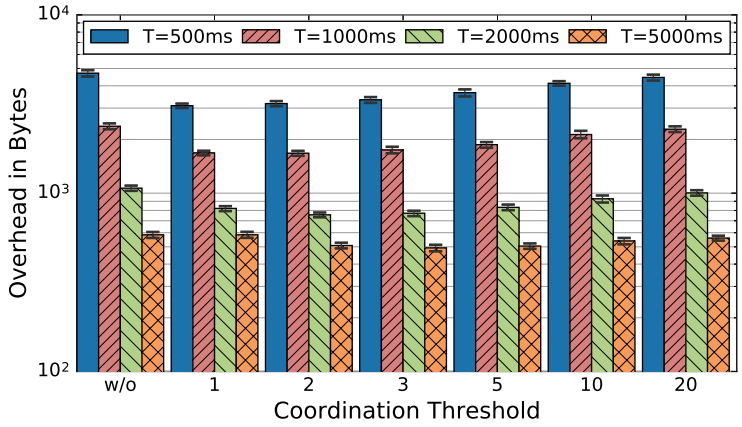


Figure 67: Comparison of the cost reduction when applying different measurement periods and different coordination thresholds using DISTTM [Har+16a].

A.4 Measurement Accuracy per Sampling Rate.

In Figure 68, we show the results of the empirical search for the most suitable probability parameters to be used in the sampling-based counter technique (SC) (cf. Section 4.1.1). On the y-axis, the figure shows the measurement error and on the x-axis the used packet sampling probability for both the ingress and egress switch. The figure shows that the accuracy is equally good for sampling rates of 2^{-2} and 2^{-4}. With lower sampling rates, the accuracy drops notably. Therefore, we select the lowest sampling rates (least costly) with the best accuracy, which is 2^{-4}.

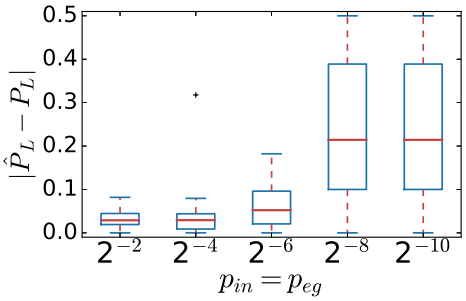


Figure 68: The error of the sampling-based counter technique depending on the sampling rate [Har+17a].

A.5 Measurement Accuracy per Measurement Update Period.

Figure 69 depicts the accuracy of the loss measurements (cf. Section 4.1.1) on the y-axis and different measurement periods on the a-axis. Different bars represent the used techniques. The figure shows that the legacy technique has almost always the best accuracy and improves the accuracy strongly with larger periods. In addition, both the active probing and in-line counter technique do not significantly change their accuracy with different measurement periods. The sampling-based counter technique shows reasonable accuracy only when the period is substantially large, in this case 180 seconds.

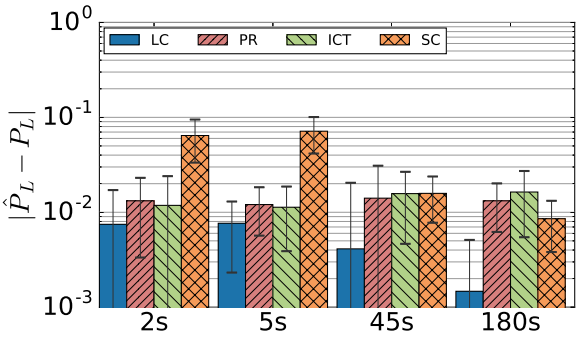
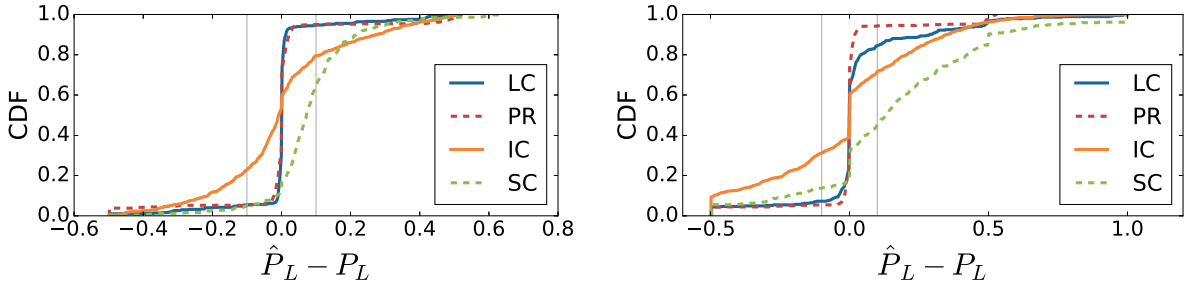


Figure 69: Comparison of the accuracy of different loss measurement techniques when applying different measurement periods [Har+17a].

A.6 Measurement Accuracy with Different Traffic Profiles.

The two subfigures of Figure 70 show the accuracy comparison for different traffic profiles when applying a loss burstiness factor of $T = 25$ (cf. Section 4.1.2). Figure 70a reveals, for the case of friendly traffic and medium-long loss bursts, that the error for all techniques is under 10% in over 60% of all cases (gray lines). For legacy counter and probing, the error is lower than 10% in roundabout 90% of the values. For bursty traffic and the same loss conditions, the accuracy decreases for the in-line counter and sampling-based counter technique. The legacy counter and active probing approaches are not affected heavily.



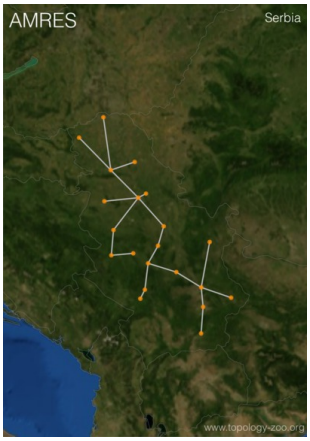
(a) Poisson traffic and Gilbert-Elliot loss profile with $T = 25$ (short-lived loss bursts).

(b) Bursty traffic and Gilbert-Elliot loss profile with $T = 25$.

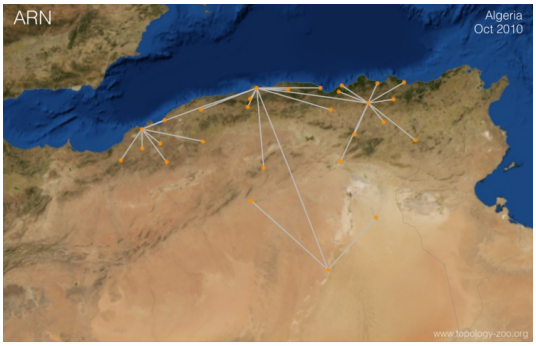
Figure 70: Accuracy of the different techniques when applying friendly Poisson and bursty traffic [Har+17a].

A.7 Illustration of used Internet Topology Zoo Topologies.

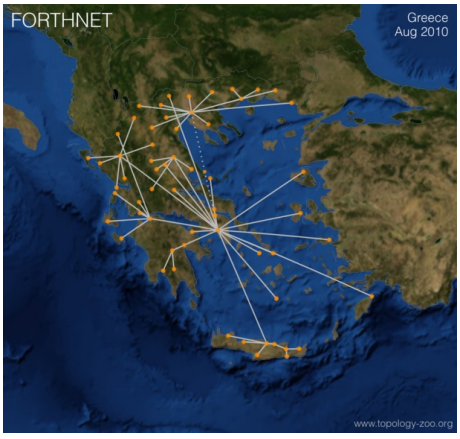
Figures 71 and 72 show the structures of the used topologies from the Internet Topology Zoo [Kni+11] on top of satellite images of the corresponding region. The depicted topologies were used to evaluate the measurement point placement based on the centrality of network nodes (cf. Section 4.2.3). The figures are taken from the Internet Topology Zoo Gallery[49] with kind permission of the publisher.



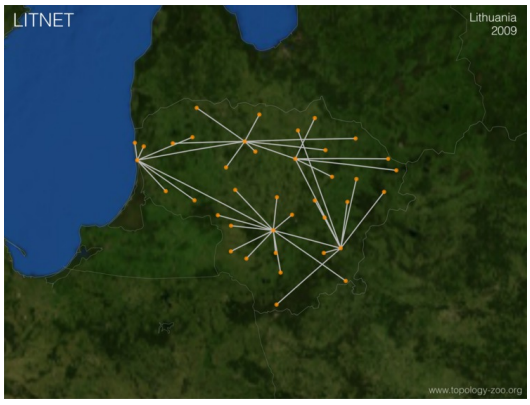
(a) The Amres topology in Serbia combines a linear and star topology.



(b) The ARN topology from 2010 in Algeria follows a tree structure with the root node located in the capital city Algiers.



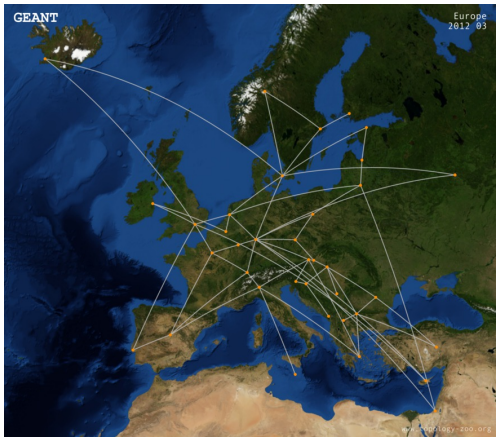
(c) The Forthnet topology from 2010 in Greece follows a tree structure with the root node located in the capital city Athens.



(d) The Litnet Topology from 2009 in Lithuania combines a ring and tree topology.

Figure 71: Star/Tree-like Topologies of Amres (Serbia), ARN (Algeria), Forthnet (Greece), and Litnet (Lithuania) [Kni+11].

49 http://www.topology-zoo.org/gallery.html, accessed 27 June 2019.



(a) The GEANT topology from 2012 spread around Europe.

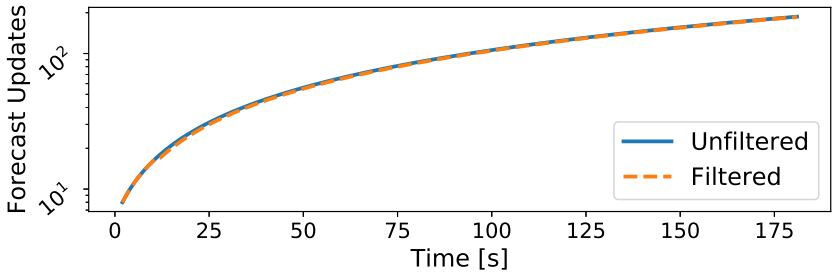
(b) The Surfnet topology from 2011 in the Netherlands.

(c) The AboveNet topology from 2011, which is spread over the world with nodes in Japan, Europe, and the USA.

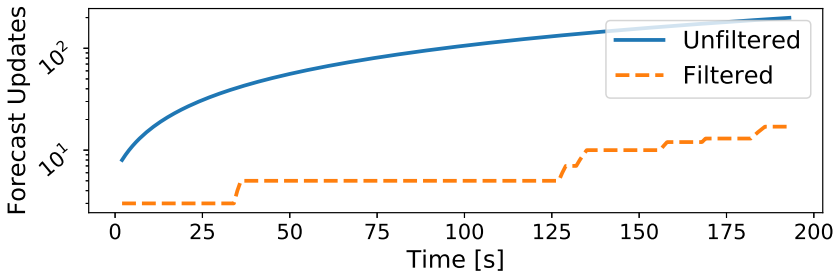
Figure 72: Meshed Topologies that do not follow a strict pattern of GEANT (Europe), Abovenet (Japan, Europe, USA), and Surfnet (Netherlands) [Kni+11].

A.8 Measurement Overhead per Improvement Threshold.

Figure 73 shows the number of forecast updates, which is equal to the number of delivered measurements to the controller over a simulation run when filtering measurement on the data-plane (cf. Section 4.3.2). In the uppermost subfigure, we see that the costs, represented by forecast updates, behave equally over the time for both cases: When filtering measurement as well as not filtering at all. This low improvement threshold of 25k leads to the case that almost all measurements pass the filter. Using higher improvement thresholds of, e.g., 500k, as given in Subfigure b, lead to the filtering of the majority of measurements. As the figure reveals, at the end of the evaluation runtime, the costs for the filtering case (dashed orange) lowers to a fraction compared to the unfiltered case (solid blue, note the logarithmic y-axis).



(a) Improvement threshold of $25k$.



(b) Improvement threshold of $500k$.

Figure 73: Costs in terms of calculated forecast updates over an exemplary evaluation run using the filtering case and without filtering [Har+19a].

A.9 Measurement Overhead and Accuracy per Delta Threshold.

The following figures show *(i)* the costs in terms of processed statistic requests and *(ii)* the corresponding error when filtering between the control-plane and data-plane (cf. Section 4.3.1). The results refer to the results given in Section 4.3.3. Figure 74 depicts the costs with changing delta threshold and without filtering at all (*Direct*). We observe the highest costs when not filtering. Furthermore, we observe a strong degradation when filtering independent of the threshold. However, for almost all thresholds, the figure shows a median of nearly zero while only the upper quartile and whiskers decrease further.

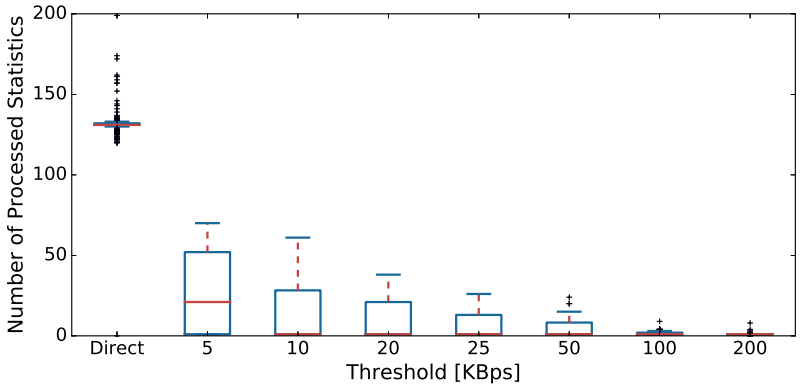


Figure 74: The costs in terms of transmitted and, therefore, processed measurements with changing delta thresholds [Har+19a].

Figure 75 shows the corresponding error when filtering between the SDN planes. Note the logarithmic y-axis. Although the difference is limited, we again observe that the upper quartile, whiskers, and outliers decrease with higher thresholds. As the threshold defines the maximum difference between consecutive measurements, the error is directly influenced.

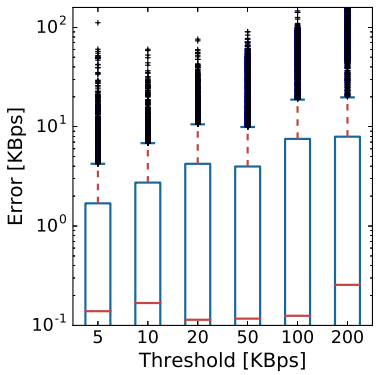


Figure 75: The error between the state in the controller and the state before filtering with changing delta thresholds [Har+19a].

A.10 Supervised Student Theses

[Ago17] Klajd Agolli. “An Element Abstraction Layer-based Monitoring Service for SDN Control Planes.” KOM-M-0585. Master Thesis. TU Darmstadt, 2017.

[Gha17] Mohamed Ghanmi. “Intelligent Selection of Measurement Points to Estimate the Flow Size Distribution in Software-Defined Networks.” KOM-B-0597. Bachelor Thesis. TU Darmstadt, 2017.

[Baw17] Anisch Bawer. “Software Defined Networking and Virtual Control Functions.” Master Thesis (*In collaboration with FG Energy Information Networks and Systems Lab (EINS)*). TU Darmstadt, 2017.

[Alb18] Dennis Albrecht. “Investigating the Dynamics of SDM Optimization Approaches in ISP Networks.” KOM-M-0606. Master Thesis (*co-supervised*). TU Darmstadt, 2018.

[Are18] Luis Felipe Villa Arenas. “Ensuring Fairness and Resilience for Flow Rule Updates in Virtualized SDNs.” KOM-M-0608. Master Thesis (*co-supervised*). TU Darmstadt, 2018.

[Zha18] Xin Zhang. “Characterizing the Packet Duplication Behavior of State-of-the-Art SDN Switches.” KOM-M-0610. Master Thesis (*co-supervised*). TU Darmstadt, 2018.

[Tou18] Ahmed Khalil Tounsi. “Decentralized Monitoring in Software-Defined Networks with Distributed Control-Planes.” KOM-B-0627. Bachelor Thesis. TU Darmstadt, 2018.

[MD19] Taushif MD. “Statistic Preprocessing to Reduce Monitoring Costs in Software-Defined Networks.” KOM-M-0657. Master Thesis. TU Darmstadt, 2019.

[Kal19] Timo Kalle. “Design and Experimental Evaluation of ”Ephemeral QUIC“ for Cyber-Physical Systems.” KOM-M-0685. Master Thesis (*ongoing*). TU Darmstadt, 2019.

B

Publications

Main Publications

[Har+19b] Rhaban Hark, Divyashri Bhat, Michael Zink, Ralf Steinmetz, and Amr Rizk. "Preprocessing Monitoring Information on the SDN Data-Plane using P4." In: *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) (accepted for publication)*. IEEE, 2019, pp. 1–6.

[Har+19c] Rhaban Hark, Khalil Tounsi, Amr Rizk, and Ralf Steinmetz. "Decentralized Collaborative Flow Monitoring in Distributed SDN Control-Planes." In: *GI/ITG Conference on Networked Systems (NetSys)*. IEEE, 2019, pp. 1–8.

[Har+18a] Rhaban Hark, Mohamed Ghanmi, Sounak Kar, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. "Representative Measurement Point Selection to Monitor Software-Defined Networks." In: *Conference on Local Computer Networks (LCN)*. IEEE, 2018, pp. 511–518.

[Har+18b] Rhaban Hark, Nieke Aerts, David Hock, Nils Richerzhagen, Amr Rizk, and Ralf Steinmetz. "Reducing the Monitoring Footprint on Controllers in Software-Defined Networks." In: *Transactions on Network and Service Management (TNSM), Special Issue on Novel Techniques for Managing Softwarized Networks* 15.4 (2018), pp. 1264–1276.

[Har+17b] Rhaban Hark, Amr Rizk, Nils Richerzhagen, Björn Richerzhagen, and Ralf Steinmetz. "Isolated In-Band Communication for Distributed SDN Controllers." In: *IFIP Networking 2017 Conference and Workshops: IFIP/TC6 Networking Poster and Demo Session*. IEEE, 2017, pp. 1–2.

[Har+17c] Rhaban Hark, Nils Richerzhagen, Björn Richerzhagen, Amr Rizk, and Ralf Steinmetz. "Towards an Adaptive Selection of Loss Estimation Techniques in Software-Defined Networks." In: *IFIP Networking 2017 Conference (NETWORKING)*. IEEE, 2017, pp. 1–9.

[Rüc+16] Julius Rückert, Jeremias Blendin, Rhaban Hark, and David 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.

[Har+16b] Rhaban Hark, Dominik Stingl, Nils Richerzhagen, Klara Nahrstedt, and Ralf Steinmetz. "DistTM: Collaborative Traffic Matrix Estimation in Distributed SDN Control Planes." In: *IFIP Networking 2016 Conference (NETWORKING)*. Ed. by IFIP. IEEE, 2016, pp. 82–90.

Additional Publications

[Kar+18] Sounak Kar, Rhaban Hark, Amr Rizk, and Ralf Steinmetz. "Towards Optimal Placement of Monitoring Units in Time-varying Networks under Centralized Control." In: *International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*. Ed. by Udo R. Krieger (Eds.) Reinhard German Kai-Steffen Hielscher. Vol. Lecture Notes in Computer Science: 10740. 19. Erlangen, Germany: Springer, 2018, pp. 99–112.

[Ric+16a] Nils Richerzhagen, Björn Richerzhagen, Rhaban Hark, Dominik Stingl, Andreas Mauthe, Alberto E. Schaeffer-Filho, Klara Nahrstedt, and Ralf Steinmetz. "Adaptive Monitoring for Mobile Networks in Challenging Environments." In: *Advances in Computer Communications and Networks: From Green, Mobile, Pervasive Networking to Big Data Computing*. Ed. by Min Song Kewei Sha Aaron Striegel. River Publishers Series in Communications, 2016.

[Ric+16b] Nils Richerzhagen, Björn Richerzhagen, Michal Lipinski, Markus Weckesser, Roland Kluge, Rhaban Hark, and Ralf Steinmetz. "Exploring Transitions in Mobile Network Monitoring in Highly Dynamic Environments." In: *Demonstrations of the Conference on Local Computer Networks (LCN Demonstrations)*. IEEE, 2016, pp. 1–3.

[Ric+16c] Björn Richerzhagen, Nils Richerzhagen, Sophie Schönherr, Rhaban Hark, and Ralf Steinmetz. "Stateless Gateways - Reducing Cellular Traffic for Event Distribution in Mobile Social Applications." In: *International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–9.

[Ric+16d] Nils Richerzhagen, Björn Richerzhagen, Rhaban Hark, Dominik Stingl, and Ralf Steinmetz. "Limiting the Footprint of Monitoring in Dynamic Scenarios through Multi-dimensional Offloading." In: *International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–9.

[Ric+16e] Björn Richerzhagen, Alexander Wagener, Nils Richerzhagen, Rhaban Hark, and Ralf Steinmetz. "A Framework for Publish/Subscribe Protocol Transitions in Mobile Crowds." In: *International Conference on Autonomous Infrastructure, Management and Security (AIMS)*. Vol. Volume 9701 of the series Lecture Notes in Computer Science. IFIP. Munich, Germany: Springer International Publishing, 2016, pp. 16–29.

[Rüc+15] Julius Rückert, Jeremias Blendin, Rhaban Hark, and David Hausheer. "DynSDM: Dynamic and Flexible Software-Defined Multicast for ISP Environments." In: *International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 117–125.

C

Curriculum Vitæ

Personal

Name	Rhaban Simon Hark
Date of Birth	21 April 1989
Place of Birth	Cologne
Nationality	German

Work Experience

Since 07/2015	Research Assistant at the Technische Universität Darmstadt, Member of the Adaptive Overlay Communications Group at the Multimedia Communications Lab

Academic Qualification

04/2012–05/2015	Master of Science: Information System Technology, Technische Universität Darmstadt Master thesis: *Dynamic and Transparently Discoverable Software-Defined Multicast in ISP Environments*
04/2014–05/2015	Master of Science: Electrical Engineering and Information Technology, Technische Universität Darmstadt
10/2008–04/2012	Bachelor of Science: Information System Technology, Technische Universität Darmstadt Bachelor thesis: *Concept and Implementation of a Framework for Content-adaption in mobile Applications*

International Experience

09/2018–11/2018	3-Month research visit at University of Massachusetts, Amherst, MA, USA, with Prof. Michael Zink at the Department of Electrical and Computer Engineering

Grants and Scholarships

2018	Recipient of the Google Scholarship for Students with Disabilities for the Academic Year 2018/2019

2018	Recipient of the German Academic Exchange Service Scholarship for Ph.D. research studies in the United States for the period of 01.09.2018 to 30.11.2018.
2018	Recipient of the Student Travel Grant of the ACM Internet Measurement Conference (IMC) 2018 in Boston, MA, USA.

Scientific Projects

04/2016–04/2019	Part of the TITAN subproject within the CELTIC-NEXT SENDATE-PLANETS project funded by the German Ministry of Education and Research (BMBF).
since 07/2015	Part of the B1 subproject within the Collaborative Research Centre 1053 *MAKI – Multi-Mechanism Adaptation for the Future Internet* funded by the *Deutsche Forschungsgemeinschaft* (DFG).

Darmstadt, July 02, 2019

Rhaban Simon Hark

D

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, 02.07.2019

Rhaban Simon Hark

Colophon

This document is based on a 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/`