Towards ML-based Management of Software-Defined Networks

Kokouvi Benoit Nougnanke

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Kokouvi Benoit NOUGNANKE

Le 12 juillet 2021

Vers un Management basé ML des Réseaux SDNs

Ecole doctorale : **SYSTEMES**

Spécialité : **Informatique**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Yann LABIT

Jury

M. Abdelhamid MELLOUK, Rapporteur
Mme Anne FLADENMULLER, Rapporteure
Mme Véronique VEQUE, Examinatrice
M. Toufik AHMED, Examinateur
M. Pascal BERTHOU, Examinateur
M. Marc BRUYERE, Examinateur
M. Yann LABIT, Directeur de thèse

Remerciements

Tout d'abord je tiens à remercier mon directeur de thèse Pr. Yann Labit, d'avoir eu confiance en moi en me confiant cette thèse de doctorat. Merci pour cette expérience très plaisante, pour les conseils et suggestions. Merci pour tout.

Je remercie tous les membres de mon jury d'avoir accepté d'évaluer ma thèse. Merci aux rapporteurs Pr. Abdelhamid MELLOUK et Dr. Anne FLADENMULLER pour le temps consacré à rapporter cette thèse ainsi que vos retours très intéressants et enrichissants. Je remercie les examinateurs Pr. Véronique VEQUE, Pr. Toufik AHMED, Dr. Pascal BERTHOU, Dr. Marc BRUYERE pour les retours et discussions intéressantes lors de la soutenance.

Je remercie mes collaborateurs Dr. Marc BRUYERE et Dr. Simone FERLIN, and Ulrich Matchi AIVODJI. J'adresse des mots de remerciements particuliers à Dr. Marc BRUYERE de m'avoir accueilli à IIJ Innovation Institute à Tokyo durant l'été 2019 et d'être resté plus qu'un collaborateur pour la suite de ma thèse.

Le LAAS-CNRS a été ma deuxième maison durant cette thèse. Je remercie son directeur Dr. Liviu NICU et tout le personnel dont particulièrement Florence à l'accueil. Je remercie les membres permanents de l'équipe SARA particulièrement Dr. Khalil DRIRA, Dr. Slim ABDELLATIF pour ses conseils précieux depuis mon stage PFE jusque lors de la préparation de la soutenance. Je remercie Dr. Philippe OWEZARSKI, également pour les discussions nourrissantes à la cantine du LAAS aux horaires tardifs.

Mes remerciements vont à tous mes collègues et amis. La liste est longue et ne la faisons pas. Merci pour vos soutiens. Merci spécial à Dr. Fadel qui a joué plein de ces rôles: ami, collègue, frère et ici aussi la liste est longue.

Je suis reconnaissant et dédie cette thèse à ma famille (NOUGNANKE et WAMPAH) pour le soutien et l'amour que vous me témoignez. Et le meilleur pour la fin, merci à toi Sandrine Eyram pour ta présence durant cette expérience, ton soutien indescriptible et ton amour.

Abstract

With the exponential growth in technology performance, the modern world has become highly connected, digitized, and diverse. Within this hyper-connected world, Communication networks or the Internet are part of our daily life and play many important roles. However, the ever-growing internet services, application, and massive traffic growth complexify networks that reach a point where traditional management functions mainly govern by human operations fail to keep the network operational.

In this context, Software-Defined Networking (SDN) emerge as a new architecture for network management. It makes networks programmable by bringing flexibility in their control and management. Even if network management is eased, it is still tricky to handle due to the continuous growth of network complexity. Management tasks remain then complex. Faced with this, the concept of self-driving networking arose. It consists of leveraging recent technological advancements and scientific innovation in Artificial Intelligence (AI)/Machine Learning (ML) with SDN. Compared to traditional management approaches using only analytic mathematical models and optimization, this new paradigm is a data-driven approach. The management operations will leverage the ML ability to exploit hidden pattern in data to create knowledge. This association SDN-AI/ML, with the promise to simplify network management, needs many challenges to be addresses. Self-driving networking or full network automation is the "Holy Grail" of this association.

In this thesis, two of the concerned challenges retain our attention. Firstly, efficient data collection with SDN, especially real-time telemetry. For this challenge, we propose COCO for **CO**nfidence-based **CO**llection, a low overhead near-real-time data collection in SDN. Data of interest is collected efficiently from the data plane to the control plane, where they are used whether by traditional management applications or machine-learning-based algorithms.

Secondly, we tackle the effectiveness of the use of machine learning to handle complex management tasks. We consider application performance optimization in data centers. We propose a machine-learning-based incast performance inference, where analytical models struggle to provide general and expert-knowledge-free performance models. With this ML-performance model, smart buffering schemes or other QoS optimization algorithms could dynamically optimize traffic performance. These ML-based management schemes are built upon SDN, leveraging its centralized global view, telemetry capabilities, and management flexibility.

The effectiveness of our efficient data collection framework and the machine-learning-based performance optimization show promising results. We expect that improved SDN monitoring with AI/ML analytics capabilities can considerably augment network management and make a big step in the self-driving network journey.

Keywords: SDN, Network management, Monitoring, Data centers, Machine Learning, Performance Optimization, Self-Driving Networks

Résumé

Avec la croissance exponentielle des performances technologiques, le monde moderne est devenu hautement connecté, numérisé et diversifié. Dans ce monde hyperconnecté, les réseaux informatiques ou Internet font partie de notre vie quotidienne et jouent de nombreux rôles importants. Cependant, la forte croissance des services et des applications Internet, ainsi que l'augmentation massive du trafic, complexifient les réseaux qui atteignent un point où les fonctions de gestion traditionnelles, principalement régies par des opérations humaines, ne parviennent pas à maintenir le réseau opérationnel.

Dans ce contexte, le Software Defined-Networking (SDN) émerge comme une nouvelle architecture pour la gestion des réseaux. Il rend les réseaux programmables en apportant de la flexibilité dans leur contrôle et leur gestion. Même si la gestion des réseaux est en partie simplifiée, elle reste délicate à cause de la croissance continue de la complexité des réseaux. Les tâches de gestion restent alors complexes. Face à ce constat, le concept de self-driving networking a vu jour. Il consiste à tirer parti des récentes avancées technologiques et l'innovation scientifique dans le domaine de l'intelligence artificielle (IA) et du machine learning (ML) en complément au SDN. Par rapport aux approches de gestion traditionnelles utilisant uniquement des modèles mathématiques analytiques et l'optimisation, ce nouveau paradigme est une approche axée sur les données. Les opérations de gestion s'appuieront sur la capacité de l'intelligence artificielle à exploiter les relations complexes et cachées dans les données pour créer des connaissances. Cette association SDN-AI/ML, avec la promesse de simplifier la gestion du réseau, nécessite de relever de nombreux défis. Le self-driving networking ou l'automatisation complète du réseau est le "Saint Graal" de cette association.

Dans cette thèse, deux des défis concernés retiennent notre attention. Dans un premier temps, la collecte efficace de données avec SDN, en particulier la télémétrie en temps réel. Pour ce défi, nous proposons COCO pour COnfidence-based COllection, une solution de collecte de données en temps quasi-réel à faible coût (overhead) pour les réseaux SDN. Les données d'intérêt sont collectées efficacement du plan de données au plan de contrôle, où elles sont utilisées par les applications de gestion traditionnelles ou par des algorithmes de machine learning.

Dans un second temps, nous explorons les possibilités de l'utilisation du machine learning pour traiter des tâches de gestion complexes. Nous considérons l'optimisation de la performance des trafics dans les data centers. Nous proposons un modèle de performance du trafic incast en utilisant le machine learning, là où les modèles analytiques peinent à fournir des modèles de performance facilement généralisables et sans des connaissances "domain-specific". Avec ce modèle de performance ML, des fonctions de management dont la gestion intelligente des switchs ou d'autres algorithmes d'optimisation de la qualité de service peuvent optimiser dynamiquement les performances des trafics. Ces opérations de gestion basées sur le ML sont construites sur une architecture SDN, en tirant parti de sa vision globale centralisée, de ses capacités de monitoring et de sa flexibilité.

L'efficacité de notre proposition de collecte de données et l'optimisation des performances basée sur le machine learning donnent des résultats prometteurs. Nous pensons que des systèmes de monitoring SDN efficaces couplés avec les opportunités offertes par l'IA/ML peuvent considérablement améliorer la gestion du réseau et faire un grand pas vers le concept du self-driving network et donc des réseaux autonomes.

Mots clés:: SDN, Management des réseaux, Monitoring, Data centers, Machine Learning, Performance, Réseaux autonomes

Contents

List of Figures

List of Tables

List of Algorithms

Glossary

AC Autonomic computing

ADN Autonomous Driving Network

AI Artificial Intelligence

API Application Programming Interfaces

AQM Active Queue Management

ARIMA AutoRegressive Integrated Moving Average

BO Bayesian Optimization

DCN Data Center Network

ICMP Internet Control Message Protocol

ICT Information and Communications Technology

IoT Internet of Things

IP Internet Protocol

LSTM Long Short-Term Memory

M-A-P-E Monitor-Analyze-Plan-Execute

MAPE Mean Absolute Percentage Error

MASE Mean Absolute Scaled Error

ML Machine Learning

NMAE Normalized Mean Absolute Error

OSI Open Systems Interconnection

QoE Quality-of-Experience

QoS Quality-of-Service

RF Random Forest

RL Reinforcement Learning

RNN Recurrent Neural Network

SD-DCN Software-Defined Data Center Network

SDN Software Defined Networking

SelfDN Self-Driving Network

SL Supervised Learning

TCP Transmission Control Protocol

USL Unsupervised Learning

WAN Wide Area Network

ZSM Zero-touch network and Service Management

Chapter 1

Introduction

1.1 Context

Communication networks or the Internet are part of our daily life and play many important roles. Computer networks play a critical role in modern society by involving in many aspects: social networks, e-commerce, business, jobs, search engines, etc. These internet services are hosted by data centers (where many servers are interconnected) connected through backbone networks (WAN - Wide Area Network). End users access these services through access networks such as WiFi, Cellular networks (4G, 5G, and 6G soon) with multiple devices (smartphone, laptops, tablets, etc.).

To caption the crucial place of the Internet, imagine what the world would be if suddenly Internet breaks. The importance of the Internet in our life is more visible with the COVID-19 pandemic. "The importance of information and communications technology (ICT) is even higher in the present crisis than usual. ICT has been crucial in keeping parts of the economy going, allowing large groups of people to work and study from home, enhancing social connectedness, providing greatly needed entertainment, etc." [Király 2020]. Other resources that highlight the importance of Computer networks during the pandemic include[1][2].

Generally speaking, the internet infrastructure survived this pandemic period successfully even if it presented some spare outages and performance (latency) degradation[3][4] [Candela 2020]. In general and particularly in such critical situations, keeping the network functioning is the role of a set of procedures, methods, and tools composing what is called network management. It allows to effectively

[1] https://www.pewresearch.org/internet/2020/04/30/53-of-americans-say-the-internet-has-been-essential-during-the-covid-19-outbreak/

[2] https://www.brookings.edu/blog/techtank/2020/04/29/covid-19-has-taught-us-the-internet-is-critical-and-needs-public-interest-oversight/

[3] https://ihr.iijlab.net/ihr/en-us/covid19?country=France

[4] https://labs.ripe.net/author/romain_fontugne/the-internet-health-report/

operate, administrate, and maintain networks by practically focusing on maintaining reliability, efficiency, and overall performance of the network infrastructure[5]. Network management includes several tasks such as monitoring, performance management, provisioning of network services, maintaining of quality of services, devices configuration, security, reliability, resilience, etc.

Indeed, the Internet is the global system of interconnected computer networks based on packet switching [Clark 1988]. For this computer to be able to communicate, they need common "languages" named as communication protocols. The protocol suite (TCP/IP) [Cerf 1974] – a conceptual model developed in the 70s – is the de facto standard used by the Internet to communicate between networks and devices. Internet is a network of networks. The network control and management follow the cybernetic approach [Wiener 1948] for controlling systems with a feedback control loop as illustrated in Figure 1.1.

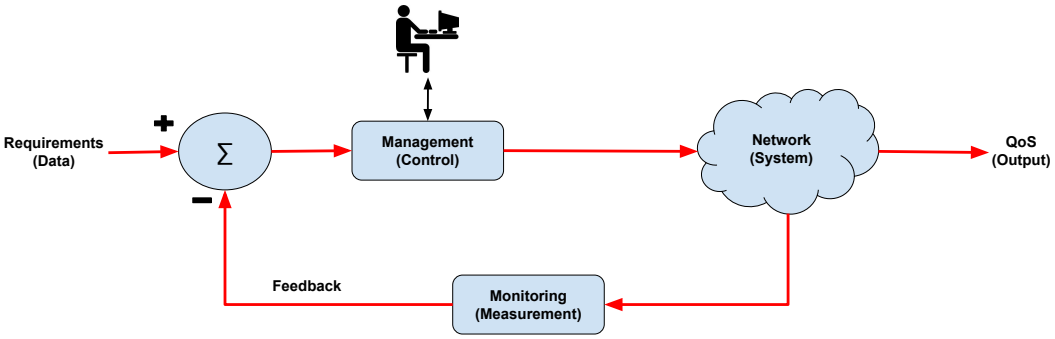

Figure 1.1: Cybernetic Feedback Loop for Network

Network management tasks are generally categorized with the so-called FCAPS functions, i.e., fault, configuration, accounting, performance, and security management [Kuklinski 2014]. Unfortunately, network management is complex and difficult. Indeed, over the years, new internet services impose new control requirements, leading to great complexity in communication systems. Control functionalities that decide how packets should be processed and forwarded constitute one of the two planes of network devices. The second one being the data-plane responsible for effectively forwarding the packets. Heterogeneous network equipment (routers, switches, middleboxes, etc .) to support thousands of new protocols ran with more than hundreds of millions of source code lines. These protocols were written using old engineering practices with poorly defined, proprietary, and closed APIs (Application Programming Interfaces) between control and data planes that hindered innovations. Network operators could not on their own experiment and implement new functionalities on their equipment. They need imperatively to pass through devices vendors, and the entire process may last several years, generally 3 to 4 years, to add a new feature. Moreover, the distributed nature of the control plane didn't ease the management. Some management tasks like configuration or

[5]https://www.extremenetworks.com/enterprise-networking/management/

monitoring were done with a centralized architecture, but this was quite limited since operating at individual protocols, mechanisms, and configurations interfaces.

Network management issues mainly governed by the growth of the network, equipment, connections, and new services, are due to several causes:

(a) Error-prone configuration (manual, in a distributed manner, individual configurations). Wide variety of different gears and management solutions.
(b) Poor troubleshooting mechanisms. Troubleshooting consists of identifying and fixing network issues as they are raised or proactively. It may suffer from a lack of global visibility, not too much automation, and advanced data analytics tools.
(c) Inefficient Monitoring. Network data collection is essential for troubleshooting, performance diagnosis, and other management tasks. But traditional monitoring tools may fail to provide good accuracy-overhead or even not suitable to collect all required information.
(d) Innovation barriers. Difficulty and even impossibility to innovate in network management (ossification of the internet) due to vertically integrated equipment with closed interfaces.
(e) Poor performance. It is caused by inefficient control and management implementations to handle the network distributed states. For example, routing and traffic engineering solutions as distributed algorithms are not efficient.
(f) Great dependence on human's ability to master complexity. For example, network performance optimization algorithms rely mostly on analytical handcrafted models. This models resulting from insights gained from empirical data and using simplification assumptions.
(g) Emerging technologies requirements. The growth of network services and new demands require more advanced communication systems for Quality-of-Service (QoS) satisfaction. We can consider here also the re-activeness against new events, unseen events or behaviours.

However, the internet infrastructure still works, thanks to network managers and administrators' ability to master complexity. Mastering complexity is not bad in itself. It is necessary for building networking systems. But when it comes to operate and manage the network, things should be easier and straightforward. In other words, complexity is needed at a low level to know exactly how things work, but simplicity must be the key at high level. Moreover, with the explosion of huge networks such as data centers, cloud infrastructures, and the large deployment of millions of IoT devices, experts mastering all the complexity may work very hard.

In software engineering and computer science, abstractions are used to build and exploit large-scale systems handling complexity efficiently. Abstractions establish a level of complexity to consider for a system for a given purpose while hiding the more complex details that are below the set level or just by hiding irrelevant details. This approach needs to be applied to computer networks, as mentioned in an iconic

talk by Scott Schenker[6].

The inventory of abstractions in networking was not that shining, especially regarding network management. Considering the two planes present in the network, the data plane worked pretty well, using layering and encapsulations as good abstractions. The OSI layering decomposes packet forwarding into fundamental components, enabling independent but compatible innovation at each layer. This made the strength of networking. But when it comes to the control plane, there were many mechanisms (routing, security, traffic engineering) pursuing their goals individually with no modularity and abstractions. Doing this way is impracticable for managing modern and future large-scale networking systems such as data centers, multi-cloud, IoT deployments, 5G/6G, etc.

In this context and with the three-fold ambition as follow *(i) enabling innovation in how networks are designed and managed*, *(ii) rethinking who is in control (or in charge) of the network* and *(iii) making networks programmable*, Software Defined Networking (SDN) arose. It separates the control and the data planes with open interfaces and provides good abstractions for the control plane [Casado 2019, Feamster 2014]. New days came for network management.

1.2 Scope and Motivations

SDN was designed as a new architecture for network design and management. SDN [Haleplidis 2015] (RFC 7426) is the physical separation of the control plane from the forwarding plane (See Figure 1.2). It eases control and introduces flexibility in network management, and more innovation in networking is eased.

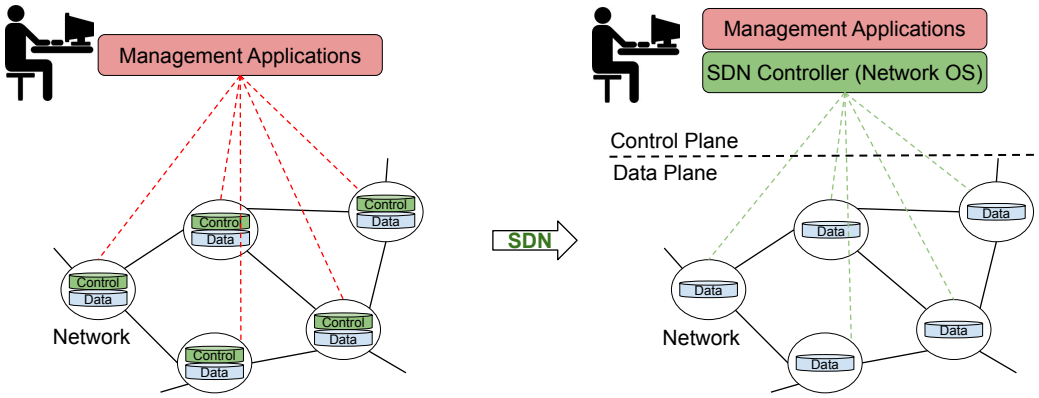

Figure 1.2: Traditional Network vs. SDN

SDN brings many advantages to network management. With the physical separation of the control plane from the data plane and logically centralized control and management, SDN handles the error-prone configuration issue and the difficulties from control algorithms operated in a distributed manner. The forwarding

[6]https://www.youtube.com/watch?v=eXsCQdshMr4

elements became simple, with open and standardized interfaces opening the way for innovation and easing their management without wondering about the underlying gears. The SDN architecture allows developing new complete monitoring and data collection frameworks. These monitoring capabilities coupled with centralized control allow great performances by abstracting the distributed states of the network conveniently. Management functions take advantage of a rich and global view of the network under control.

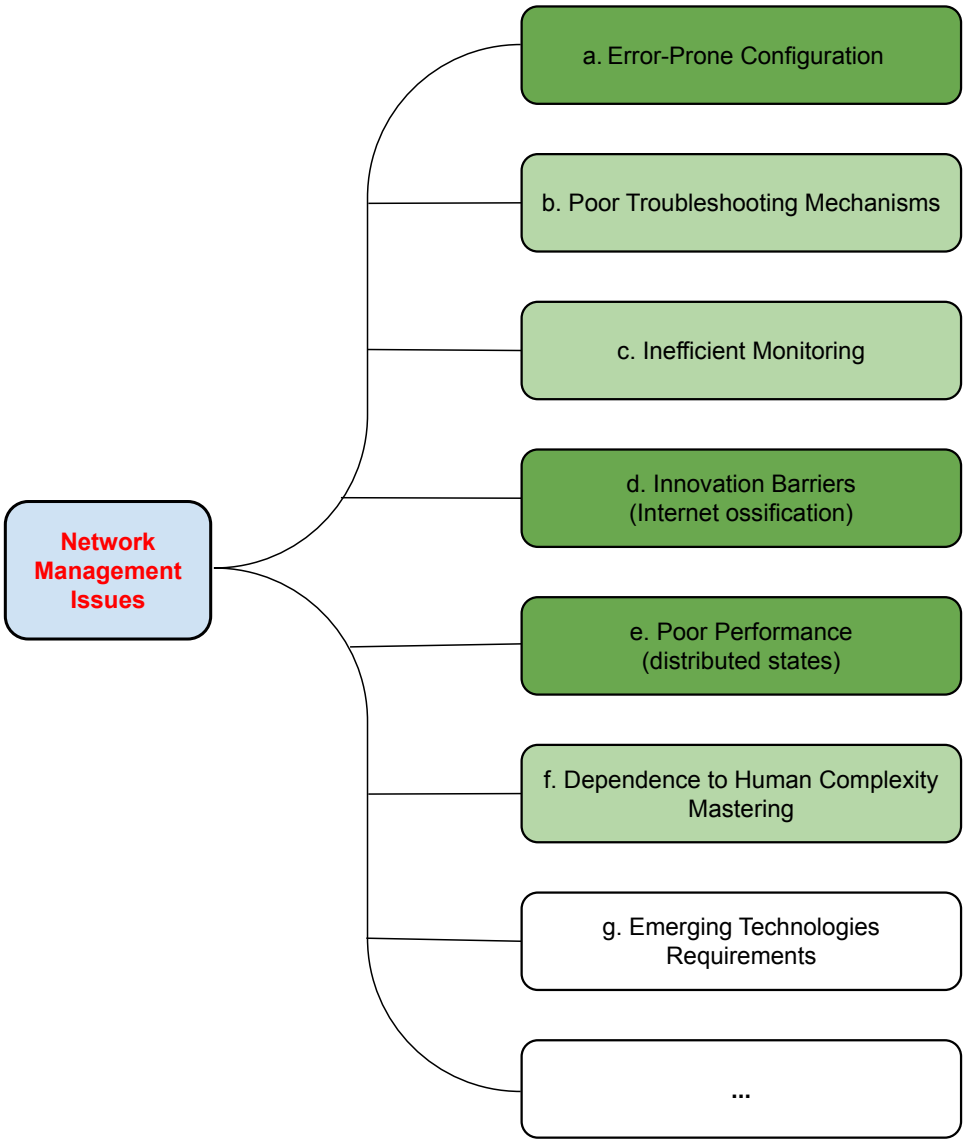

Figure 1.3: Network Management Problems Handled by SDN

SDN enables addressing some of the management issues discussed above as illustrated in Figure 1.3 (dark green: well addressed; light green: partially addressed; white: not addressed). With SDN, we have more resilience even if the single point of

failure problem is raised. Additionally, with SDN, we have the new issue of control plane/data plane communication overheads. Moreover, With SDN, new interesting services and platforms have emerged (e.g., multi-cloud Datacenters) that come with new complex problems. Thus, SDN doesn't resolve all the management problems, even if it eases many things, and more it comes with its own challenges.

Nevertheless, SDN presents good advantages and a promising future for networking. Then, adopting SDN and identifying the remaining challenges could ensure an efficient network management scheme. In other words, with SDN, network management reaches an interesting level. Still, new investigations need to be done to allow network management to be more efficient and be able to handle modern and future networks[7]. Then, despite the significant advantages of SDN, it is neither a complete solution for network management problems nor free of problematic issues:
Need for Improved SDN Monitoring. As an illustration, Although the separation of the control plane from the forwarding plane comes with a lot of advantages (flexibility, programmability, open interfaces, etc.), communications between the data plane and the SDN controller generate significant overhead. This overhead concerns firstly control and configuration by flow rules installation, packet-in/packet-out handling, and on the other hand, the network information collection. It's then important to design efficient monitoring schemes with good accuracy-overhead trade-off.
Need for More Automation. The increased autonomy with SDN is suitable but not sufficient regarding the ever-growing complexity of the network that has to accommodate heterogeneous workloads with various performance requirements. Indeed, even with SDN, the human operator's role in network management is very significant, even for tasks related to "real-time"/"near-real-time" decision-making. This approach may scale terribly in people costs. And as if that wasn't enough, most network faults are caused by human manual error. It is then necessary to research more and more autonomy in management tasks, and Artificial Intelligence (AI)/Machine Learning (ML) can help.

Therefore, ensure great performance of networks with the rapidly evolving applications services, even with SDN, becomes challenging with the classical control loops involving human intervention or dependence. Then, inspired by autonomous vehicles, self-driving networks vision which leverages SDN and ML, gained significant attention [Feamster 2017, Kalmbach 2018].

The self-driving network (SelfDN) concept or Autonomous Driving Network (ADN) or Zero-touch network and Service Management (ZSM) [Feamster 2017, Huawei 2020, ETSI-ZSM 2019] is an emerging trend faced the network that becomes more and more complex and seemingly unmanageable. Indeed, the main objective of network management teams is to continuously deliver application and service performance and protection for users. Then, they need continuous network monitoring and optimization to support increasingly dynamic, digitally-driven busi-

[7] `https://www.cisco.com/c/dam/m/en_us/solutions/enterprise-networks/networking-report/files/GLBL-ENG_NB-06_0_NA_RPT_PDF_MOFU-no-NetworkingTrendsReport-NB_rpten018612_5.pdf`

ness models, which is not the case with SDN alone or ML alone applied to network problems. This prowess could be achieved by leveraging AI/ML for optimization on top of SDN that will provide a convenient environment with monitoring capabilities, data-driven decision-making, machine-learning-based automatic optimization, adaptive control, and flexibility. Literally, with this approach, networks would learn to drive themselves, opposed to human rule-based operations that rely on closed-form analysis of individual protocols and components.

With SelfDN, the management plane can get precisely the relevant data from the network to train ML models that will drive actions. It's worth pointing out that it doesn't mean that humans are entirely removed from the whole management loop. Humans are still needed for some tasks (verification, certain constraints definition, etc.). But some complex management tasks could be automated, at least partially.

As mentioned in [Feamster 2017], the networking research community is already constructing the building blocs of SelfDN. The puzzle has to be gathered efficiently within the SelfDN adventure. Indeed, the trip has begun a long in the past with the concept of autonomic networking that follows the concept of autonomic computing introduced by IBM in 2001 [Horn 2001]. Autonomic computing (AC) aims to develop systems with self-x properties such as self-configuration, self-healing, self-optimizing, self-protection, etc. to overcome the rapidly growing complexity of computing systems management to reduce the barrier that complexity poses to further growth [Kephart 2003].

The AC architecture is represented by the M-A-P-E Monitor-Analyze-Plan-Execute (M-A-P-E) formalism that uses closed control loop as a basis [Ayoubi 2018]. The M-A-P-E functions interact with a knowledge source. The functions can retrieve data from this source and also add new insights (See Figure 1.4).

Classical management engineering, tightly coupled with human expert knowledge, fails to come up with practical autonomic self-driving networks. This failure is due to several reasons: the lack of great visibility on the network, barriers of distributed control, vendor lock-in of legacy networking devices, the lack of programmability and flexibility that would allow adaptively configure and operate the network elements autonomously. Fortunately, SDN fills those gaps and providing new chances for autonomic networking. More, by observing that the cognition was restricted to the analyze function only within the original M-A-P-E, the authors of [Ayoubi 2018] revisited M-A-P-E and propose the cognitive M-A-P-E control loop (C-M-A-P-E) for network management. C-M-A-P-E integrates cognition for all the functions (C-Monitor, C-analyze, C-Plan, C-Execute). It allows each function to operate in total autonomy. Recent ML advances are also of great importance. ML then helps achieve cognition through learning and inference, as shown in Figure 1.4.

In line with the SelfDN vision, the C-M-A-P-E aims to minimize the role of human operators in the control loop because human error is the main cause of outages. THe C-M-A-P-E enforces the classical control loop from Figure 1.1 to achieve intelligent automation using ML. Indeed the network complexity is taught to have been grown far beyond the limits of manual administration. A very close paradigm is the Knowledge-Defined Networking [Mestres 2017] that restates D. Clark et al.'s

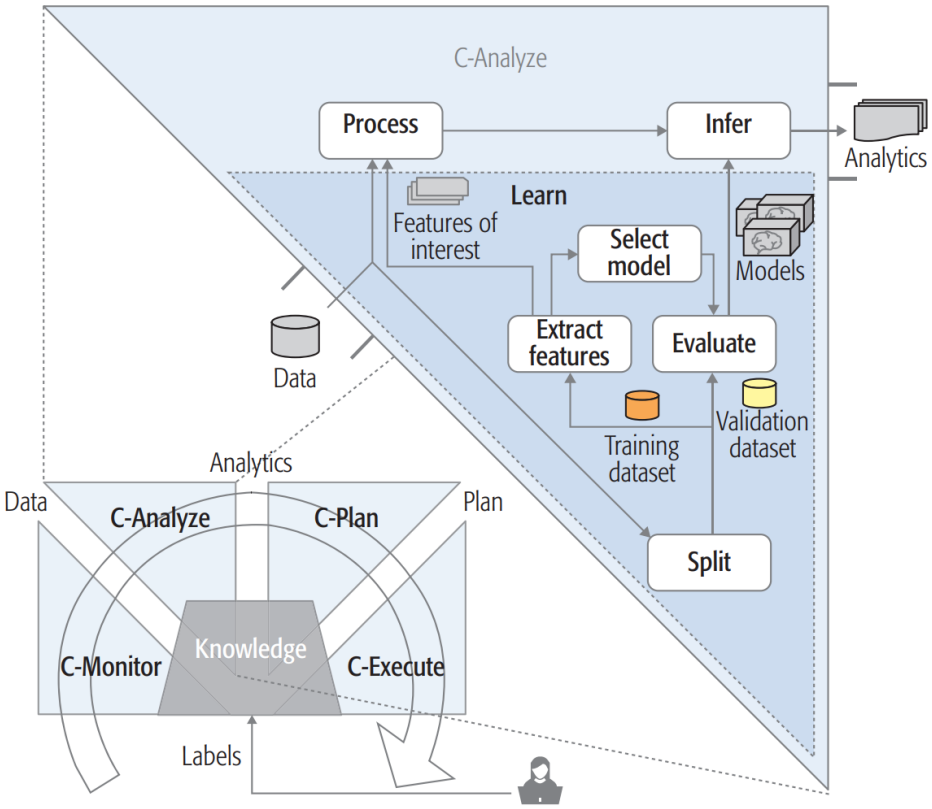

Figure 1.4: Cognitive control loop for network management (from [Ayoubi 2018]).

proposed "A Knowledge Plane for the Internet" [Clark 2003] in the context of SDN-based management.

With advances in SDN, recently data-plane programmability with P4, advances in AI/ML in hardware, algorithms, and toolkit democratization (Scikit-Learn, Keras, Tensorflow, Pytorch, etc.) leveraged with these autonomic formalisms, we expect a promising future for SelfDN.

1.3 Thesis Statement and Contributions

In this thesis, we argue that:

« By designing efficient monitoring for SDN and applying unprecedented capabilities offered by ML and AI for complex network management tasks, we can considerably improve network management and make a big step toward full network automation then self-driving networks. »

Network telemetry or monitoring that consists of collecting and analyzing measurements to understand what is happening in the current network and support its management operations is not only crucial for network management. But, it is also a key enabler to let networks run themselves in the context of the vision of the self-driving network that leverages the C-M-A-P-E control loop with SDN. Our first contribution deals with efficient telemetry in SDN architectures. For management tasks automation with SDN, we consider the use case of data centers, a key element in today's Internet, that presents challenging tasks where AI/ML need to be investigated. Our second contribution then uses ML for performance prediction in SDN-enabled Data Center Networks (DCNs). Finally, the third contribution leverages the ML performance modeling capabilities for performance optimization.

Towards advancing the State-of-the-Art, this thesis makes these three contributions as follow:

Improved SDN Telemetry. SDN's promised global view needs to be up-to-date for efficient and near-optimal control and management, which means near-real-time network state observation. Inspired by monitoring solutions proposed for SDN architectures in the literature and new monitoring trends, we propose a monitoring framework for the construction of the global and up-to-date view of the network data plane in SDN. It makes a clear distinction between information that needs to be collected in an ad-hoc manner with classical OpenFlow polling techniques, events that need to be conveyed by notifications, and periodic statistics (especially counters) collection. The only downside is that the periodic collection may generate a lot of overhead for an SDN architecture. To handle this overhead, i.e., provide timely accurate statistics to the control plane with low overheads, adaptive pull-based approaches were proposed. These approaches consist of using adaptive polling rates instead of a fixed one. These state-of-the-art approaches provide good accuracy-overhead trade-offs, but we thought we could do better by detaching from the classical OpenFlow request-reply model for the particular case of periodic statistics collection. We then propose COCO (**CO**nfidence based adaptive **CO**llection),

a push and prediction based adaptive periodic OpenFlow statistics collection mechanism that decreases the overhead considerably in terms of the number of messages used to collect this flow statistic in near-real-time, with almost no accuracy degradation.

ML Incast Performance Inference in SD-DCN. Modern cloud-native applications (big data analytics, IP storage, etc.) bring new traffic patterns on DCNs, including incast where multiple servers communicate simultaneously with a single client. The dynamic DCN workload with heterogeneous QoS requirements (high throughput for elephant traffic and low completion time for incast) complexify data center management tasks (e.g., congestion control, buffer management, QoS optimization algorithms). These performance optimization algorithms need performance models providing insights on how various factors affect the performance metrics (classical management workflow: "measure-model-control(actions)"). The existing models for incast are analytical models that are either tightly coupled with a particular protocol version or specific to certain empirical data. Motivated by this observation, we propose a machine-learning-based incast performance modeling engine capable of learning from collected historical data and predicting incast performance metrics. The learning approach has the advantage of being independent of underlying protocols and any restricted assumptions. The power of data is leveraged to achieve this prowess. The ML model can capture complex relationships from diverse system parameters to accurately predict application performance in DCNs.

Automatic ML and Bayesian Optimization (BO) based smart switch buffer management in SD-DCN. Switch buffer management consists of finding the best operating parameters such as switch buffer space and AQM parameter values to achieve efficient performance for applications. Choosing the right buffer size and Active Queue Management (AQM) parameters for optimal performance is challenging due to the complexity and dynamics of the data center environment. More, these parameters interact in complex ways, and then it isn't easy to develop analytical performance models that can guide the choices. We are then in the presence of a difficult optimization problem where the objective function (performance model) is unknown. Thanks to Bayesian Optimization (BO), a powerful tool designed to solve such problems where the objective has no simple closed-form. BO supposes, however, that this performance function can be evaluated at any parameter combinations. But this is not possible for online decision-making by the adaptive smart buffering scheme. And it is at this point that we leverage the ML performance model developed in the second contribution. This performance model builds upon historical data of past parameter configurations, and their observed performance can provide such evaluations through predictions.

The takeaways of these introductory sections are presented with the thesis big picture as illustrated in Figure A.2. It highlights our view of the Internet as described earlier and the challenges concerned with this thesis.

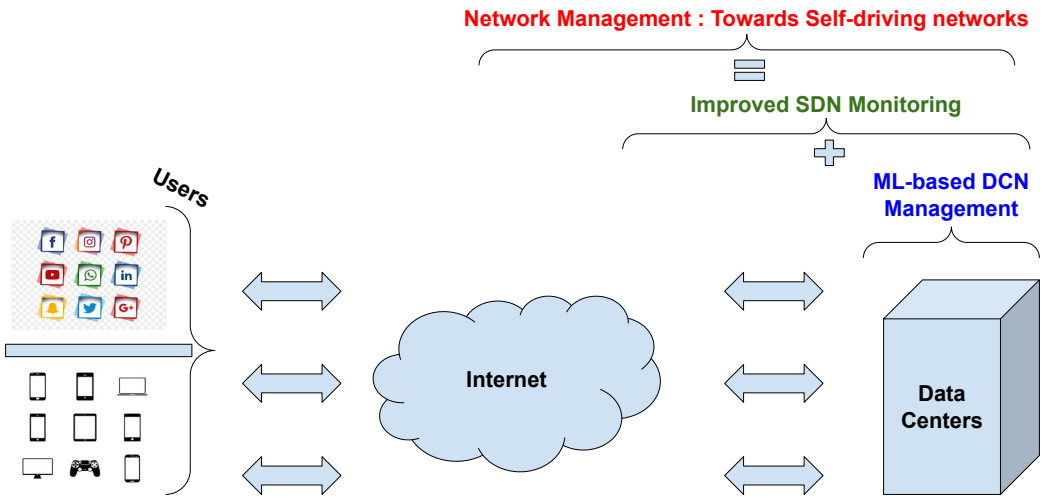

Figure 1.5: Thesis Big Picture

1.4 Dissertation Organization

The whole thesis is structured as shown on Figure 1.6. The body of the dissertation is composed of two parts handling the two main challenges identified within SDN (Section 1.2) in modern complex networks and that goes in the same direction as the self-driving networks vision: *the need for improved SDN monitoring* and *the need for more automation*.

Part I - Efficient network telemetry in SDN. It is composed of two chapters.

Chapter 2 presents a general overview of network telemetry and its challenges within traditional network management. It introduces the SDN architecture and how it eases network management and then monitoring. The importance of data collection for SDN architecture functioning is also discussed. The chapter ends by presenting the overhead problem of periodic data collection in SDN.

Chapter 3 first presents a data collection framework for SDN telemetry. The framework comprises a set of components, including a periodic collection overhead handling element. For this element, we propose COCO, a push and prediction-based adaptive algorithm for continuous flow statistic collection. Intensive evaluation results using Mininet and real traces show that our proposition can achieve up to 75% overhead reduction compared to a fixed pushing collection with almost no accuracy degradation.

Part II - AI/ML for complex management tasks in SDN-enabled DCNs to handle the need for more autonomy for management operations. It includes three chapters.

Chapter 4 provides a general overview of data centers, how they work, and their role in the actual internet. It introduces the incast and elephant management problem in DCNs. We expect ML to handle the complexity related to the traffic dynamics and heterogeneous performance requirements. This chapter provides a

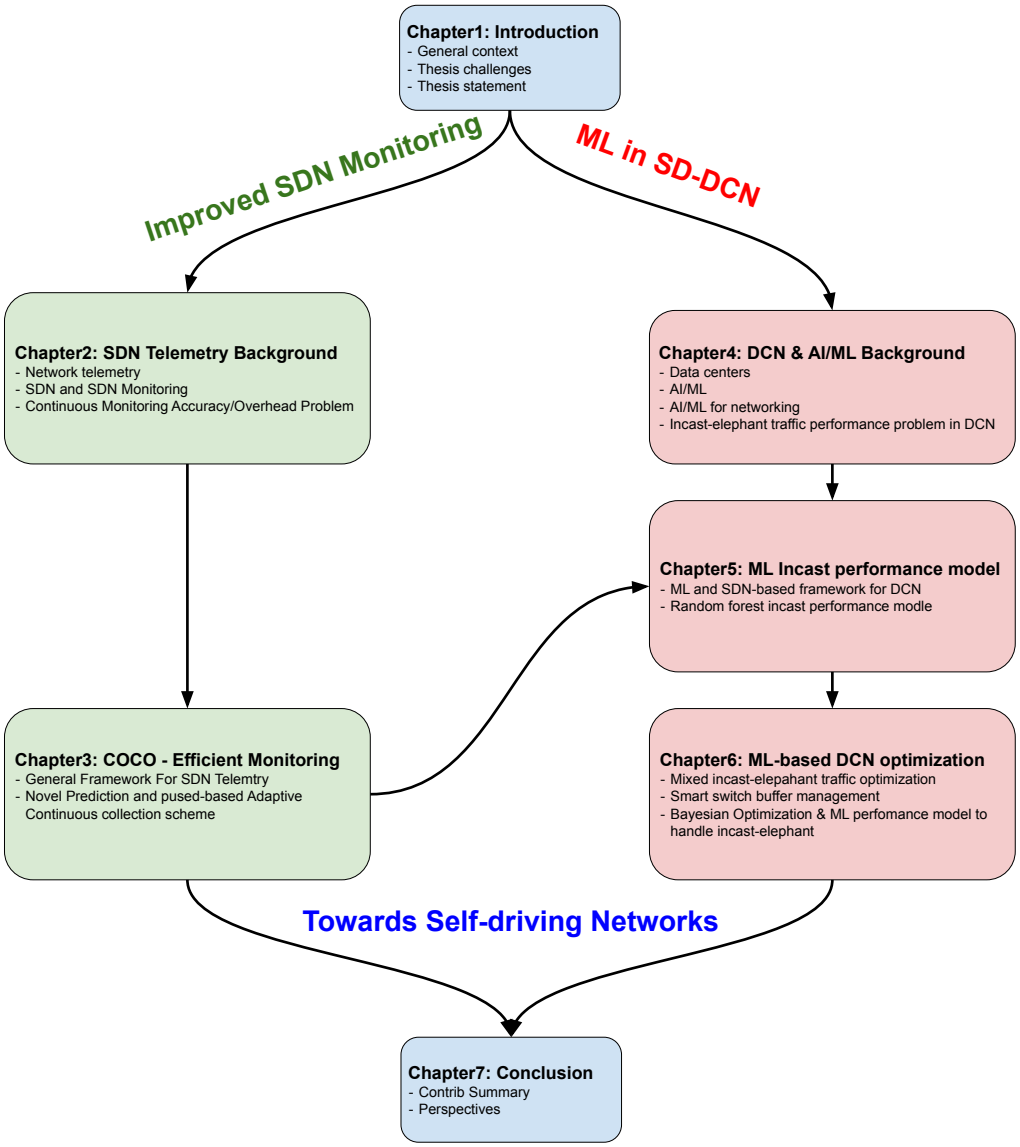

Figure 1.6: Thesis Organization

brief background on AI/ML, a literature review of its use for network problems (security, performance, QoS, etc.), and the potential of the hybridization of ML and SDN as enabling technology for modern and future networks management. The chapter ends by presenting a base architecture leveraging SDN and ML in a control loop to handle mixed incast and elephant traffic in DCNs.

Chapter 5 presents an ML framework for incast performance (completion time) modeling instead of analytical models that depends on empirical data or simplifications assumptions, but that generalize poorly. We use a Random forest that captures complex relationships from several parameters (the number of incast senders, the bandwidth of bottleneck link, etc.) and infers the corresponding incast completing time. We discuss some challenges within the ML approach, especially prediction latency.

Chapter 6 presents how we leverage the ML performance model for incast and elephant performance optimization. We focus on switch buffer management by formulating an optimization problem to find the best parameters (buffer space, AQM parameters) to provide optimal performance (minimal incast completion time and maximal elephant throughput). The resulted optimization problem is a black-box optimization problem for which we propose Bayesian optimization and the ML inference for its resolution. This approach allows automatic and adaptive switch buffering to provide efficient performance in the DCN dynamic environment.

Finally, **Chapter 7** concludes this thesis and outlines future research directions.

Part I

SDN Monitoring

Chapter 2

SDN Monitoring Preliminaries

Network telemetry (or monitoring) is an important element of network management that aims to determine the behaviour of the network and the status of its elements (e.g., hosts and switches). It helps making a variety of management decisions for improving the performance and QoS. However, it is challenging to build real-time and fine-grained network monitoring systems because of the need to support a variety of measurement queries, handle a large amount of traffic for large networks, while providing a good accuracy/overhead trade-off. Monitoring overhead concern the resource constraints at hosts and switches and network bandwidth utilization.

This chapter, organized in three sections, aims at proving the general context needed for our first contribution concerning SDN monitoring. First, Section 2.1 provides an overview of network telemetry by presenting definitions, traditional network monitoring tools, and their challenges with traditional network management mainly governed by manual operations. Then Section 2.2 as a new architecture for network management with the goal of brings for agility, flexibility, and an amount of automation in network management. We present data collection mechanisms within SDN before ending the chapter with the periodic data collection overhead problem in Section 2.3.

2.1 Network Telemetry 101

2.1.1 Overview

Network telemetry or monitoring consists of *collecting* and *analyzing* measurements *to understand what is happening* in the current network to support its management

operations[1]. The questions network administrator may care about are, for example: "Is there a congestion in the network?", "What are the largest flows of in the network?", "Is there any performance anomaly?" etc. The collected information and insights gained on the network would guide control and management decisions concerning traffic classification, anomaly fixes, performance improvements, troubleshooting, reliability, only to mention this. Then, network telemetry is not only crucial for network management. It is also a key enabler *to let networks run themselves* in the context of the self-driving network vision as illustrated with the C-MAPE architecture in Figure 1.4, where management and telemetry are in a closed control loop.

Traditionally, **network monitoring** forms with **network design** and **network deployment** a cycle of network operations [Lee 2014]. Network monitoring thus consists of collecting and reporting information from collection devices to a central management station. It's worth pointing out that, even if network operators performed control operations (e.g., routing) and device configurations in a distributed manner, some management operations like monitoring were conducted with a centralized approach. The collected data helps infer the network behavior, which is analyzed and guides the design of configuration changes. These configurations and infrastructure changes are then deployed on the network, which will continue to be monitored. And the design-deploy-monitor cycle continues.

Network telemetry with the aim to build a model of the network's current behavior involves resource monitoring and performance monitoring. Besides this classification, there are endpoint-based and switch-based monitoring. The former is more close to applications but lacks network visibility. At the same time, the latter allows having a broad range of information leveraging its more wide visibility, but it lacks direct information on application performance. Finally, depending on the methods used to collect and transmit the telemetry data from those measurement devices, the monitoring can be active (using probes, generally on end-system) or passive (with no additional monitoring actions on the devices)[2][3] [Labit 2014].

Active monitoring consists of transmitting probes into the network and collect metrics-of-interest based on the test packet or their response. They are used to determine the end-to-end network performance as it is experienced by the applications. The common active management tools include *ping* and *traceroute*. Ping allows measuring delay using ICMP Echo Request and Echo Reply messages. Traceroute allows determining the network topology by following the path of a packet by identifying the routers through which it passes. Iperf is also an active tool used to measure delay, jitter, and estimate bandwidth. With the probes, actives methods generate overhead on the network and interfere with normal traffic on the network. Unlike active monitoring, passive monitoring does not inject packets into the network. It collects at a specific point in the network, for instance, a switch. The collected information (captured packets, statistics) is post-processed and analyzed

[1] https://www.youtube.com/watch?v=VzbV0ceRAVU
[2] https://www.caida.org/catalog/datasets/monitors/
[3] https://www.cse.wustl.edu/ jain/cse567-06/ftp/net_monitoring/index.html

to obtain information about the network behavior. For active monitoring tools, we may distinguish hardware solutions (Endace DAG) and software ones (tcpdump, Wireshark, NetFlow, etc.). Our contribution on SDN monitoring falls into the category of passive methods, where we collect statistics from a network switch.

To be effective for network management, network monitoring must follow the following requirements: (i) *High accuracy* to support optimal actions; (ii) *High resource-efficiency* meaning with a minimal overhead (e.g. memory, computation resources); (iii) *high scalability* to support a high numbers of flows, devices; (iv) *high generality/flexibility*; (v) *etc.*

2.1.2 Traditional Monitoring and Challenges

Traditional network telemetry solutions include SNMP, NetFlow, IPFIX, sFlow, Ping (ICMP), Tcpdump, Wireshark, etc., [So-In 2009, Zhou 2018]. Let us describe some of these tools. Netflow is a Cisco solution using probes attached to switches that collect either complete or sampled flow statistics and send them to a central controller. The standardized version of NetFlow is IPFIX for IP Flow Information Export from the IETF working group. sFlow, for "sampled flow", is a sampling approach that uses a time-based sampling approach to capture traffic information. Others types of solutions include local traffic information retrieving as packet sniffers such as Tcpdump and Wireshark. And we will finish with the most used tool, SNMP (Simple Network Management Protocol), a management protocol composed of a set of IETF standards for network data collection from network devices. It can also be used for configuration by modifying information maintained by SNMP agents on devices to change their behavior.

One of the main characteristics of traditional monitoring is related to the design-deploy-monitor cycle properties that are executed mainly by manual or human operations. And configuration errors are one of the most causes of misconfiguration [Mahajan 2002]. These error-prone configurations may require much troubleshooting and monitoring from operators. Then, improving design operations can reduce the time to monitor the network. In this context, some improvement works propose solutions in configuration verification and simplification [Feldmann 2001, Feamster 2005]. The operations simplification can be achieved through the automation of certain tasks. For this automation to be effective and widely adopted, it needs standard interfaces and data formats, which was not the case.

Moreover, new trends in networking (IoT, 5G, cloud-based data centers, etc.) create complex and highly dynamic environments where traditional monitoring techniques (SNMP, NetFlow, sFlow, etc.) are in trouble to achieve the above requirements. SDN monitoring approach is a possible improvements [Tsai 2018].

2.2 SDN Architecture and its Telemetry Capabilities

2.2.1 Software Defined Networking

SDN [Haleplidis 2015] (RFC 7426) was designed as a new architecture for network design and management with the physical separation of the control plane from the forwarding plane. These three points guided the SDN proposition (i) *Enabling innovation in how networks are designed and managed*; (ii) *rethinking who is in control (or in charge) of the network (e.g., equipment vendors? network operators?)*; (iii) *Making network programmable.*

The OSI (Open Systems Interconnection) model (see Figure 2.1[4]) developed in 1983 is a stacked protocol model composed of 7 layers from the physical to the application layer used as a basis for network design. This layering decomposes packet forwarding into fundamental components and then presents good abstractions for the data-plane. These abstractions enable independent but compatible innovation at each layer[5]. Over the years, the network growth and new control requirements led to great complexity. Indeed, network operators used highly complex equipment (e.g., routers) created to support more than 100 hundreds of protocols (IETF RFCs and IEEE standards) with more than 100 million of source code based on old engineering practices and using poorly defined APIs. These networking gears were intended to serve many customers as one product for the benefit of the manufacturers. This complexity – mastered by network managers and administrators to keep the network working – was questioned and pointed to the need for control planes abstractions as the key of control and management operations simplification. It was then difficult to operate networks and difficult to innovate in networking, causing what is called *the internet ossification.*

With the simplification purpose and the difficult innovation in networking based generally on a vertically integrated design approach based with vendor locking, early SDN researchers propose to bring computing and software engineering approach to networking. Indeed the computing industry, historically composed of vertically integrated mainframes, was transformed into a horizontal marketplace with open interfaces and multiple options at every level. This enables innovation at every level, such as operating system one (with Linux, Windows, and macOS on top of microprocessor chips). On the application level, there was a considerable growth of applications[6].This approach to networking consists of bringing **three abstractions:** (distribution abstraction, forwarding abstraction, and configuration abstraction) and **two open interfaces** (to rely the 3 planes as in Figure 2.2) in how network are implemented and managed [Feamster 2014, Casado 2019].

The distribution abstractions aim to provide a global view that hidden the distributed state of the network. Centralized algorithms can then be run on this global view. For instance, concerning routing, the Dijkstra algorithm can be run

[4]https://commons.wikimedia.org/wiki/File:Osi-model-jb.svg
[5]https://www.youtube.com/watch?v=eXsCQdshMr4
[6]https://sdn.systemsapproach.org/index.html

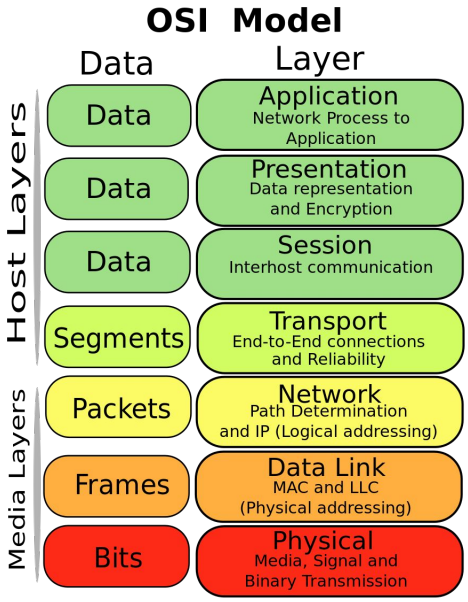

Figure 2.1: The OSI Reference Model

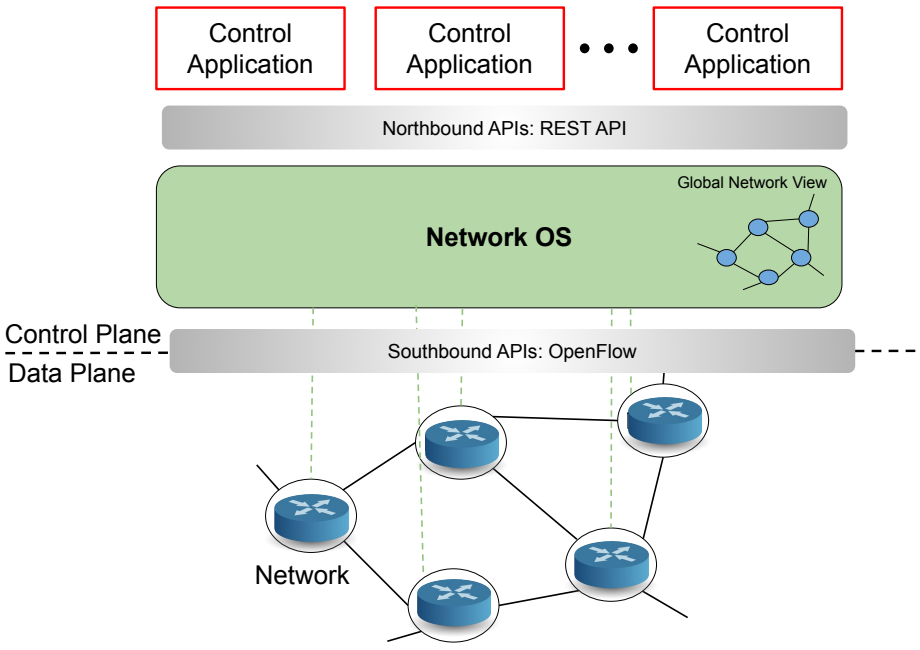

Figure 2.2: SDN architecture: Abstractions & Open Interfaces

on the network graph instead of using the Bellman-Ford algorithm. This approach has the merit to make a distributed control problem a logical centralized one. The distribution abstraction involves designing a common distributed layer, the so-called network OS (see Figure 2.2). The network OS's role is to gather information from the network elements and disseminate control and configuration commands to the network elements.

The forwarding abstraction hides details of the underlying hardware. It then provides a flexible model for the control plane. One of the widely used abstraction is the OpenFlow "Match+Action" abstraction. Configuration abstraction aims to provide agnostic and model-based configuration of network devices. Examples of configuration models are YANG, OpenConfig, etc. All these abstractions enable then to have simpler forwarding devices.

OpenFlow [McKeown 2008] is a fundamental building block of SDN. It is the first open standard interface separating the network and data planes. This open API allowed the forwarding plane elements to be externally configured and controlled by an SDN control. OpenFlow is then one of the open interfaces proposed between forwarding elements and the network OS. It's categorized as southbound API. The other interface is the northbound API, between the network OS and the control applications. Figure 2.3 presents the main component of OpenFlow-enabled switches that compose the SDN data plane.

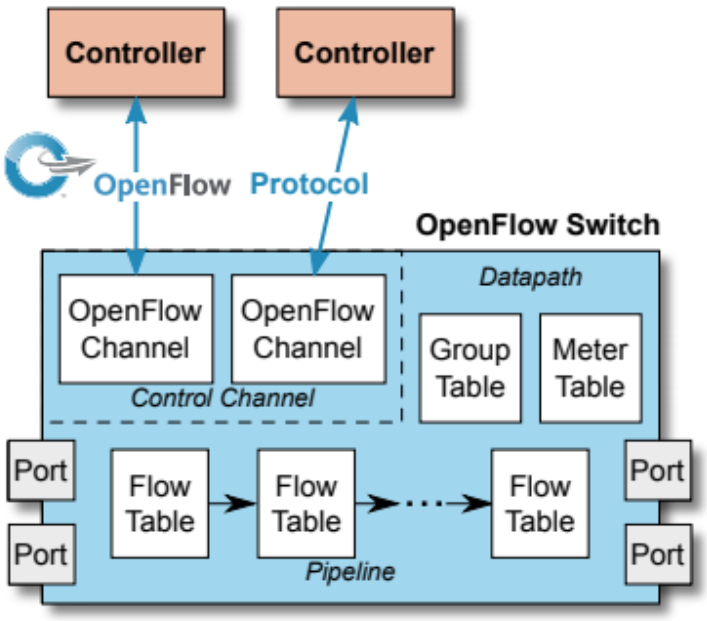

Figure 2.3: The main Components of an openFlow switch (from [Found 2015])

To sum up, SDN is a network architecture that decouples the network control and forwarding functions where a single control plane controls several forwarding devices. The SDN architecture presents several interesting characteristics among

which it is agile, centrally managed, directly programmable, programmatically configures, its is based on open standards and is vendor-neutral[7].

2.2.2 SDN Architecture

SDN controller is the heart, not to say the brain of an SDN architecture [ONF TR-521]. It exercises management and control over the data-plane devices to satisfy the needs of applications by translating their requirement (high-level intent) into flow rules (instruction) that will be executed by devices [ONF TR-516]. When satisfying these requirements, the controller has to achieve specific goals: QoS provisioning, optimal network resources utilization, security aspects, etc. The SDN controller ensures the control part essentially with the OpenFlow protocol [Version]. For management, protocols used are OF-CONFIG, NETCONF, SNMP, etc.

By controlling and managing the underlying data plane, the SDN controller plays the role of an active control element in a feedback loop, responding to network events, recovering from failure, reallocating resources [ONF TR-502], etc. Because the data plane is a dynamic environment that changes over time, the controller, by mediating between applications and network resources [Schaller 2017], continually adapts the state of resources of its data-plane to maintain a desired (optimal) state.

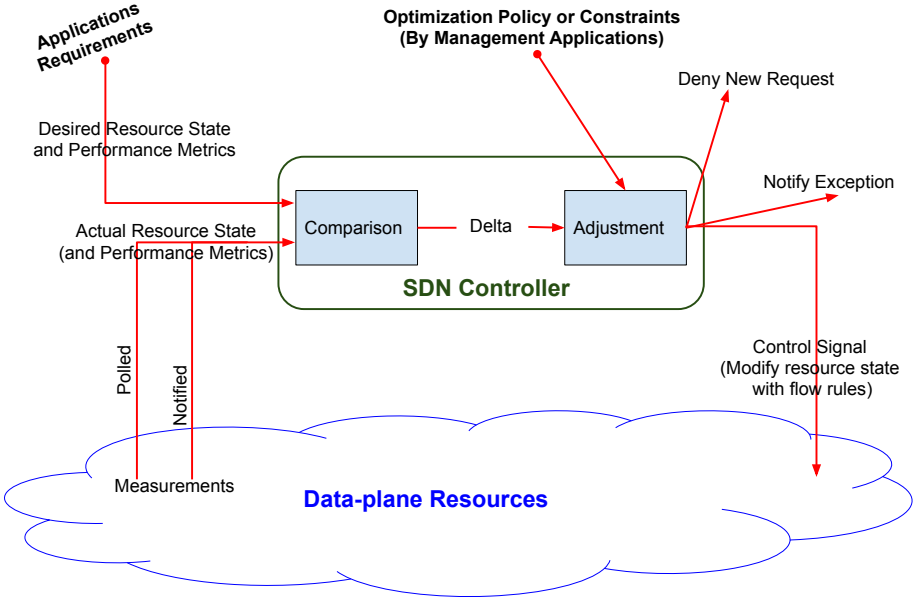

Figure 2.4: SDN Controller in a Control feedback loop

From Figure 2.4, adapted from [ONF TR-521] [Schaller 2017], we can see control as the process of establishing and maintaining a desired state on the data-plane, and feedback is of utmost importance in this process. The feedback process constitutes a management task (events handling, statistics collection, monitoring, etc.).

[7] https://opennetworking.org/sdn-definition/

2.2.3 Data collection in SDN

As shown above, with an SDN architecture, we are in the presence of a closed control loop. For this control loop to be efficient (i.e., allows applications to dynamically control the network while improving its utilization, QoS provisioning, etc.), the most active element of this loop, the SDN controller, has to provide to the applications network information pertinent to their needs and objectives [ONF TR-516]. We can distinguish two main categories of information:

- **Traffic Forwarding information**: flow statistics such as switch interfaces (ports) information like flow volume, data throughput, packet loss, port down or up, etc.
- **Network Resource State information**: related more to the management part e.g., CPU usage, disk usage, etc.

The information may be directly informative (port is down or up) or can be used as performance metrics and so on. As mentioned by ONF in SDN Scope and Requirements this information that can be static (eg. datapath ID) or dynamic (eg. CPU usage) may be collected in several manners: **provisioned**, **discovered**, **queried**, **notified** and **streamed**.

Using these techniques, the SDN controller would gather relevant information and provide a global view and full knowledge of the data-plane under its control. The data collection importance is not to be proved. Still, it is increasing with the advent of machine learning techniques in networking in general [Boutaba 2018] and especially in the control loop of SDN [Xie 2018], machine learning who breathes "only" by data. Monitoring will also have a great role in the implementation of self-driving networks. Self-driving networks need continuous resource monitoring, and streaming telemetry fits well with this requirement [Rafique 2018]. Nevertheless, care should be taken to the abstract level and granularity of collected information and to the cost of this task.

Indeed, Streaming Telemetry[8], powered by the OpenConfig working group, is a new approach for network monitoring in which data is streamed from devices continuously. Thus bulk time series data is collected (e.g., statistics are uploaded every 15 seconds). An event-driven update and request/reply mechanism for ad-hoc data are also available. Unlike SNMP, Streaming Telemetry uses a publish-subscribe pattern and is a model-driven telemetry scheme enabling vendor-agnostic monitoring. It is based on open-source initiatives like the YANG model, OpenConfig data model, gRPC (Google Remote Procedure Call), etc.

2.3 Periodic Monitoring High Overhead Problem

For efficient and near-optimal control and management with SDN, the global view of the network needs to be up-to-date, which means near-real-time network state ob-

[8] http://www.openconfig.net/

servation. One practical way to do this is by collecting non-event-based data (traffic counters especially) periodically at high frequency (e.g., reports interval of few seconds). Continuous monitoring is then ineluctable in SDN, and this monitoring must not be restricted to only traffic or flow information but all relevant ones that can be useful in any decision-making process at the controller level. The only downside is that it may generate a lot of overhead for an SDN architecture. To handle this issue, adaptive data collection techniques are developed. Based on [Tsai 2018], an overview of network monitoring in SDN, [Tangari 2018] an adaptive approach for monitoring in SDN and [Taherizadeh 2018], presenting a state-of-the-art review of monitoring in edge computing and its requirement, the literature adaptive techniques handle efficiently overhead reduction problem, but they still present these three aspects:

- they are tailored for a polled-based monitoring model and generally for traffic and flow information collection. Other information may be useful, and for continuous monitoring, a push model is more adequate as in streaming telemetry. So, adaptive techniques have to be compatible with this latter model or more have to be generic;
- they require complex parameters tuning, which limits their adoption;
- the lightness of the proposed solutions for source nodes that need to dedicate most of their processing capabilities to forwarding tasks is questionable.

The network state information collection, mainly done periodically, is guided by several reasons. For example, we want network state information to be available when SDN services or applications request them, a proactive approach for the network management. This periodic data collection is not without cost, but more it presents a dilemma: *at which frequency (sampling period), the data have to be collected*?

- If we use a high frequency, we will have real-time and accurate information but with significant overhead.
- A low frequency resolves the problem of overhead, but we will not have enough accurate network state information to support management operations.

Continuous monitoring, not to say periodic monitoring at high frequency, is then needed to ensure high-visibility and deep or real-time insights on the data plane. Streaming telemetry (discussed above) has arisen and is compatible with these requirements by streaming data each X seconds to an interested receiver, on a push basis, generating bulk time series data. Concerning the collection overhead, the use of a push model offers better efficiency compared to a collection in a polling fashion. Furthermore, to add more efficiency and tackle this overhead problem, we propose to stream the data, not with high frequency, but with a **"virtual high frequency"**, or in other words, to use an adaptive sampling in the data streaming process. The adaptive sampling procedure is based on a confidence index α representing the

ability of the receiver, here the controller, to predict missing data points according to the polling frequency adjustment. Our adaptive monitoring algorithm called COCO will be presented in more detail in Chapter 3. Figure A.2 highlights the concerns and contributions of this first part of the thesis concerning preparing SDN for augmented and more intelligent management through its monitoring capabilities improvement.

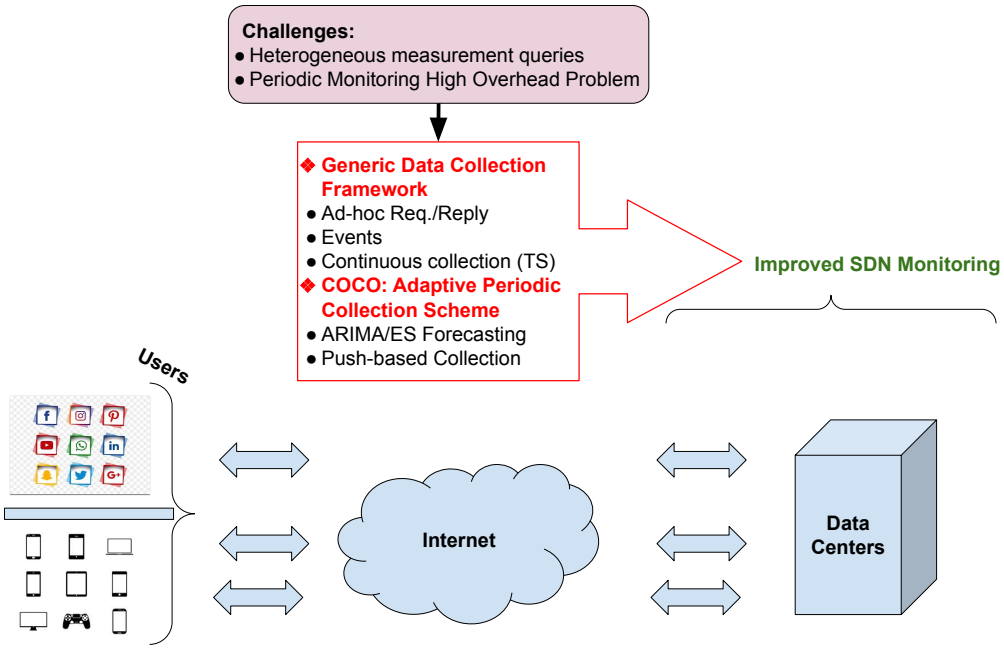

Figure 2.5: Thesis Part I: Improved SDN Telemetry

Chapter 3

COCO: Confidence based Collection - Efficient Monitoring

In this chapter, we present an architecture for network data collection in SDN. The near-real-time collection of flow-level statistics provided by OpenFlow (e.g., byte count) needed for control and management applications like traffic engineering, heavy hitters detection, attack detection generate a lot of overhead. Indeed, this near-real-time collection is conducted with periodic polling at high frequency. Periodic polling may generate high overheads expressed as the number of OpenFlow request and reply messages on the control network. To handle these overheads, we propose a push and prediction-based adaptive collection to handle efficiently periodic OpenFlow statistics collection while maintaining good accuracy. We utilize the Ryu Controller and Mininet to implement our solution, and then we carry out intensive experiments using real-world traces. The results show that our proposed

approach can reduce the number of pushed messages up to 75% compared to a fixed periodic collection with a very good accuracy represented by a collection error of less than 0.5%.

This chapter is structured as follows. First, Section 3.1 present the detail of our proposed monitoring framework. Then in Section 3.2 we provide a the problem of periodic collection of OpenFlow switches statistics. After what a we present the problem formulation. Section 3.3 presents the adaptive collection mechanism (COCO) that decreases considerably the overhead in terms of number of messages used to collect flow statistic in near-real-time. We implement the proposed algorithm using the Ryu Controller and Mininet, and after that, we carry out intensive experiments using real-world backbone and university data-center traces. The evaluations results are presented and analyzed in section 5.5. We also present in this section a direct use ace of flow counter collection. It consists network utilization (bandwidth) monitoring, essential for traffic engineering. We discuss the scalability aspect of our proposition and position it with SDN new directions as P4 and INT in section 3.5. Section 3.6 presents the related works. Finally, Section 6.7 concludes the chapter.

3.1 Monitoring Data Collection Framework for SDN

SDN control applications like making optimal routing decisions under certain QoS constraints and management applications need flow-level real-time monitoring. The use of traditional monitoring techniques (SNMP, IPFIX, Netflow, sFlow) is not encouraged due to scalability concerns; they may require special instrumentation of the network (switches) or may impose significant overheads on the underlying system. In this way researchers proposed specific monitoring frameworks for SDN environment among which [Chowdhury 2014, Yu 2013, Hartung 2017] to mention a few. They are designed with scalability, flexibility, and especially lightweightiness(frameworks with low overhead) concerns. These frameworks are composed of blocs like Request Interpreter, Scheduler, Switch Selector, Aggregator, Databases, Counters blocs, Metrics computation bloc, Restful APIs, etc. All these frameworks took advantage of OpenFlow flow-level statistics (number of packets or bytes matching a specific flow, flow duration, etc.), counters, and event-messages such as Packet-In, Flow Removed messages for the monitoring task.

Recently, new monitoring paradigms have been proposed in networking, including In-band Network Telemetry (INT) and Streaming Telemetry. In this part of the thesis, we focus on streaming telemetry[1], powered by the OpenConfig working group. It is a new approach for network monitoring in which data is streamed from devices continuously. Thus bulk time series data is collected (e.g., statistics are uploaded every 5 seconds). An event-driven update and request/reply for ad hoc data are also available. Unlike SNMP, Streaming Telemetry uses a publish-subscribe pattern and is model-driven telemetry enabling vendor-agnostic monitoring. It is

[1] http://www.openconfig.net/

based on open-source initiatives like the YANG model, OpenConfig data model, gRPC (Google Remote Procedure Call), etc.

Inspired by streaming telemetry, we leverage the monitoring frameworks discussed above and come up with the following data plane state data collection framework for SDN, presented in Figure 3.1.

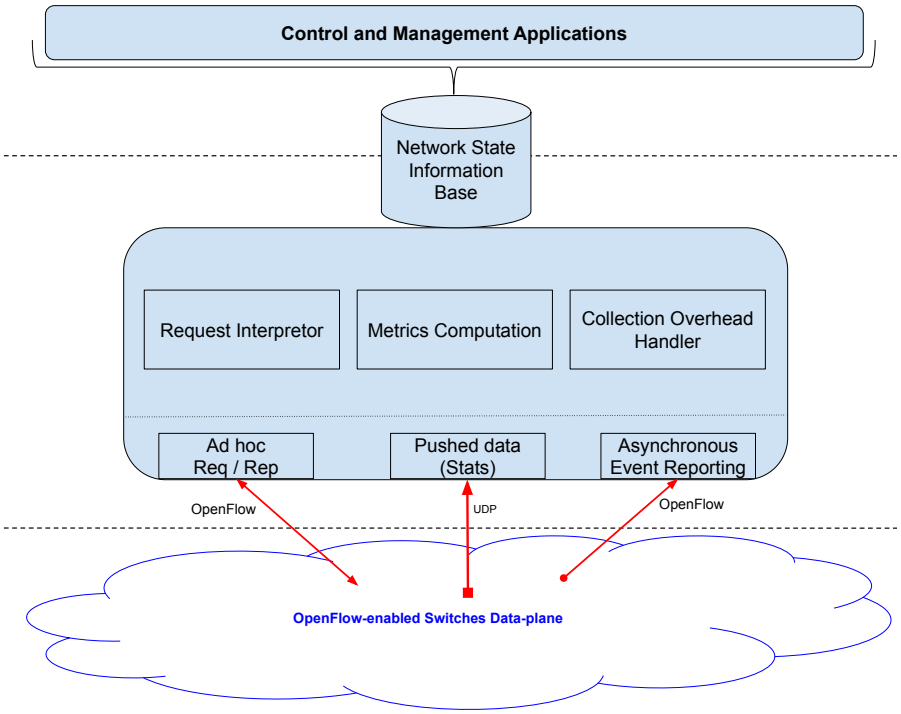

Figure 3.1: Monitoring Framework

The framework is composed of **Network State Information Base** that stores collected information. A **Request Interpretor** translates high-level monitoring intents and requirements from control and management services as statistics collection, which is done continuously with a push model over UDP or punctually with OpenFlow request-reply messages, and also with event-based reporting. A **Metrics Computation** bloc provides QoS and performance metrics based on collected statistics (e.g., bandwidth estimation through OpenFlow bytes counts time series). And finally, we have a **Collection Overhead Handler**, responsible for providing the global and up-to-date view at low cost, especially when it comes to continuous (periodic) statistics pushing.

3.2 Periodic Data Collection Problem

3.2.1 Preamble

Continuous monitoring, not to say periodic monitoring at high frequency, is needed to ensure high-visibility and deep or real-time insights on the data plane. The

SDN controller will leverage this information to control the network through the OpenFlow API. Apart from being used to control the forwarding plane through flow rules installed by the control plane, OpenFlow exposes flow statistics (packets and bytes count, flow duration, etc.) and other relevant flow information (flow entry expiration and packet-In). These statistics and event-based information need to be gathered to enriched the global view of the network and be used by many control and management functions as traffic engineering [Curtis 2011], routing [Akin 2019], anomaly detection [Zhang 2013], congestion detection, identification of architectural bottlenecks, etc. This flow-level measurement is done either actively, by querying the statistics such as traffic counters, flow duration on a pull basis with Read-State messages, or passively after an event as flow entry expiration and packet-In.

Moreover, for efficient and near-optimal control and management, the global view of the network needs to be up-to-date. This means near-real-time network state observation. One practical way to do this is by collecting non-event-based data (traffic counters especially) periodically at high frequency (e.g., reports interval of few seconds). But OpenFlow's pull-based statistics retrieving may impose high overheads [Curtis 2011], expressed by the number of request and reply messages on the data plane.

To handle these overheads, i.e., provide timely accurate statistics to the control plane with low overheads, pull-based adaptive approaches using adaptive polling rates instead of a fixed polling frequency were proposed. The collection granularity adjustment is achieved in two different ways. Either according to the stability of the statistic being collected, by comparing the difference between two consecutive data-points to one or several thresholds [Chowdhury 2014]. Or the adaptiveness uses a prediction-based approach which adjusts the reporting intervals according to the ability to provide good estimates of future data points based on the historical set composed of the previously collected data points [Zhang 2013, Tangari 2018].

The approaches mentioned above provide quite good accuracy-overhead trade-offs. But we can do better by detaching from the classical OpenFlow request-reply model for the particular case of periodic statistics collection. We then propose a push and prediction-based adaptive periodic OpenFlow statistics collection with low overheads and almost no accuracy degradation. This will be possible since the switch – the source of the statistic being collected – is involved in the adaptive process. Then it automatically knows the adaptive collection points in time, and on its own, it will just push the statistic value controller without the need for request messages. Doing this way, the reduction of overhead will be more consequent.

Before jumping to the problem formulation section, it's worth point out the importance of *counters*, the statics of interest of this study. Indeed, packet switches maintain statistics for several reasons such as performance monitoring, network management, network tracing, traffic engineering, and security. The statistics are usually collected by counters which might, for example, count the number of arrivals of a specific type of packet or count particular events, such as when a packet is dropped. The arrival of a packet may lead to different statistics counters being updated [Shah 2001]. In SDN, where there is a clear separation between the control

plane and the forwarding plane, these counters originating in the forwarding plane need to be conveyed to the control plane. Moreover, the data-plane switches use a match-action paradigm, and the forwarding of packets is done by matching flow rules that generate bytes and packet counts. The packet bytes counter is very useful for management applications like heavy hitter detection [Harrison 2018], volumetric attacks detection [Hamza 2019], link bandwidth estimation, etc.

3.2.2 Problem Formulation

A flow statistic S to be collected in near-real-time, here OpenFlow bytes count on data plane switch, is modeled by:

$$S : \mathbb{R}^+ \longrightarrow \mathbb{N}$$
$$\forall u, t \in \mathbb{R}^+ : u \leq t \implies S(u) \leq S(t)$$

S(t), representing the bytes count matching a specific flow, or a certain port bytes count in the interval [0, t), is a cumulative, non-negative and non-decreasing function. This information must be part of the global and up-to-date view exposed by the control plane. It may be used to compute some performance metrics like network utilization, a key metric for management and control functions like Traffic Engineering. Practically the statistic will be collected periodically, and these functions' requirements define the period T_0 to be used. And then the statistic S may be seen as: $S_{T_0} = \{s_i\}_{i=0}^{n}$ n $\to \infty$, a large sequence of data-points s_i at regular time interval T_0 with $s_i \leq s_{i+k}, \forall k \in \mathbb{N}$, representing an increasing trend time series. We denote the collected version at the controller level as $\hat{S}$. If collected at a fixed T_0, $\hat{S_{T_0}}$ will be considered equal to S_{T_0} (with the hypothesis that there is no packet loss and that data plane and controller communication delay is zero, ideally with an out-of-band control deployment).

For fine-grained near real-time control and management, T_0 needs to be as small as possible, but this may generate a lot of overhead. A question that could raise at this point is: *"Isn't any magic T_0 to be used that will give a good trade-off between the collection accuracy and the overhead ?"*

3.2.3 Pushed and Predicted based Adaptive Data Collection

Our solution consists of not to have to look for this magic T_0. Instead, we will provide the sample of S(t) with the application chosen T_0, but not all the samples will be really collected. We effectively collect at adaptive frequency (T_0, $2T_0$, $3T_0$, ...) and the non collected data-points will be predicted based on the already exiting ones. In other words with our adaptive data collection, $\hat{S_{T_0}} = \{\hat{s_i}\}_{i=0}^{n}$ where some data-points are really collected ($\hat{s_i} = s_i$) and the other are predicted based on the previous ones. Not like in the fixed collection mechanism where $\hat{S_{T_0}} = S_{T_0}$, here $\hat{S_{T_0}} = S_{T_0} + \varepsilon$, where ε is the collection (prediction) error.

One other important optimization opportunity concerns the OpenFlow Statistic retrieving model that uses a request-reply mechanism. It's obvious to notice that in a periodic collection or adaptive collection like in our case, the collection points are known in time by the source node (the data plane switch) after the collection initialization. Then the request message is of any importance. The node has to push the information at the collection points on its own.

We propose our Pushed and Predicted based Adaptive Data Collection mechanism respecting the following requirements:

- Lightweight solution for data plane elements since their main role is forwarding. Heavyweight tasks (e.g., prediction) are delegated to the collector in the control plane. Indeed imposing extra functions on the switches may increase the processing time of packets.
- It may not require complex parameters tuning, essential to enable easy adoption and evolution.
- The solution must reduce the number of messages compared to a fixed push mechanism with a bounded error when a fraction of inaccuracy is tolerable.
- And other requirements such as it must be simple to deploy, may not suffer from packet loss, etc.

3.3 COCO: Periodic Statistic Collection Overhead Handling

3.3.1 Overview

We are not the first to address this problem of period collection overhead reduction. With existing solutions, the impact is more visible in terms of data-plane switches and controller loads. Still, the overhead related to the dissemination (communication) of monitoring messages is not too much treated, and our monitoring optimization module takes this seriously into account. When proposing a push model, we divide by two (2) the number of messages to be disseminated compared to a pull mechanism because we do not need at all request messages to be sent from the controller to the switches before having statistics information.

Researchers were pessimistic on a push model for monitoring data collection in SDN, saying that it may require "new instrumentation" on switches. Faced this, we propose the easy to deploy, transparent and lightweight solution that uses an adaptive algorithm based on a confide index for collecting traffic counters, statistics presenting a particular pattern, an increasing trend.

The Collection Overhead Handler bloc (from Figure 3.1), as we will see later, enables reducing the periodically pushed data overhead considerably while maintaining very good accuracy. It is built upon two mainstays: the first one is **the adaptive pushing** and the second one is **the power of time series forecasting**

(prediction). By adaptive push, unlike fixed push where to get $\hat{S_{T_0}}$ at the controller level, $\hat{s_i}$ will be pushed by the data plane switch at regular T_0 time interval, to get $\hat{S_{T_0}}$, $\hat{s_i}$ will be pushed at adaptive frequency T_0, $2T_0$, $3T_0$ ... and the missing $\hat{s_j}$ according to T_0 granularity, will be predicted, based on previous collected $\hat{s_i}$. The controller level's ability to provide good estimations of uncollected data points will guide the adjustment of the adaptive frequencies.

Indeed, we represent these adaptive frequencies or periods as $T_\alpha = (1 + \alpha)T_0$, where α is a confidence index expressing a degree of confidence between the collector and the node, source of the statistic, on the ability of the prediction method to provide good estimations of uncollected data-points. The confidence index α is constructed incrementally. This means that if for a given α the predictions are good, we pass to a new confidence level $\alpha + 1$. Conversely, the confidence index will degrade when we do not have good estimates. Hence our push and prediction-based adaptive collection solution is called COCO for **CO**nfidence based adaptive **CO**llection.

Architecturally, our collection overhead reducing mechanism operates on two entities: the collector and the data plane agent, as shown in Figure 3.2. We develop two algorithms, one per entity working in collaboration. The algorithms are presented in the next section.

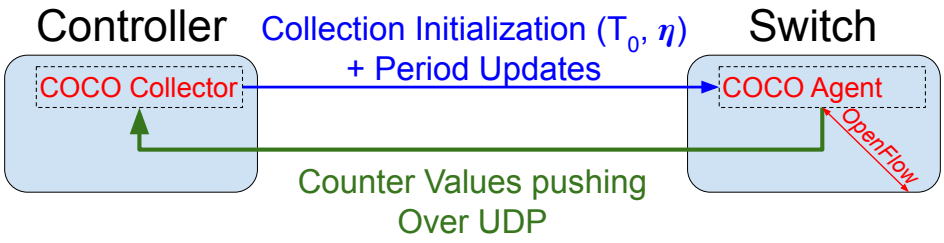

Figure 3.2: COCO Collection Architecture

3.3.2 COCO Algorithms

The COCO Agent executes the adaptive push procedure (see Algorithm 1) as a software-based monitoring solution on the node (switch), of course in collaboration with the SDN controller as mentioned earlier. In fact, for a statistic that is required to be up-to-date at T_0 granularity at the controller level, we attach the confidence index α, and $T_\alpha = (1+\alpha)T_0$ is the period with which this information (the samples $\hat{s_i} = s_i$) will be pushed. T_α will evolve in a similar manner as TCP's congestion window evolution: slow-start and congestion avoidance. Indeed, we will begin with a low period $T_\alpha = (1+\alpha)T_0 = T_0$ ($\alpha = 0$) and α will be increased each β data-points pushed if a deviation alarm is not raised by the controller. Otherwise, α will be decreased according to the deviation degree γ.

The process on the controller side is described with Algorithm 2, where the collector processes collected data-points ($\hat{s_i}$), forecasts uncollected ones ($\hat{s_j}$) and

Algorithm 1: COCO Agent: Adaptive Pushing Procedure

Push β_0 data-points $\hat{s_i}$ each T_0, $(\alpha = 0)$;
$\alpha \longleftarrow \alpha + 1$ and $T_\alpha \longleftarrow (1 + \alpha)T_0$;
$i \longleftarrow i + 1$;
while *Forever* **do**
  Push $\hat{s_i}$ at T_α ;
  if *($\hat{s_i}$ is the β ith data-point for α)* **then**
    if $\neg$ *Deviation occurred* **then**
      $\alpha \longleftarrow \min(\alpha + 1, \alpha_{max})$;
    else
      $\alpha \longleftarrow \alpha$ / (2 * γ) ;
    end
    $T_\alpha \longleftarrow (1 + \alpha)T_0$;
  end
  $i \longleftarrow i + 1$;
end

Algorithm 2: COCO Collector: Forecasting and Deviation Checking

Collect the first β_0 data-points $\hat{s_i}$;
$\alpha \longleftarrow \alpha + 1$ and *steps* $\longleftarrow \alpha + 1$;
while *Forever* **do**
  Collect data-point $\hat{s_i}$;
  Add $\hat{s_i}$ to last_collect ;
  if *($\hat{s_i}$ is the β ith data-point for α)* **then**
    if *error_indicator(last_collect, last_forecast)* $< \eta$ **then**
      No Deviation: $\alpha \longleftarrow \min(\alpha+1, \alpha_{max})$;
    else
      Deviation γ: $\alpha \longleftarrow \alpha$ / (2 * γ) ;
      Send Asynchronously γ to the switch;
    end
    steps $\longleftarrow \alpha + 1$;
    last_collect $\longleftarrow \emptyset$;
    last_forecast $\longleftarrow \emptyset$;
  end
  Forecast *steps* future data-points ($\hat{s_j}$) using previous $\hat{s_i}$;
  Add the *steps* ith forecast to last_forecast ;
end

raises deviation alarm if needed.

The controller receives samples $\hat{s_i}$ from the node each $T_\alpha = (1 + \alpha)T_0$. Firstly it forecasts *step* equals to $\alpha + 1$ next data-points. The first α forecasts will be used as estimation of $\{\hat{s_j}\}_{j=1}^{\alpha}$ uncollected data-points between $\hat{s_i}$ and $\hat{s_{i+1}}$ on a T_0 basis.

Secondly, The last forecast the $\alpha + 1$ ith one will be compared to $\hat{s_{i+1}}$ to estimate the prediction algorithm accuracy. For a given α, after collecting β data-points $\hat{s_i}$, an accuracy deviation checking is done using these data-points $\{\hat{s_i}\}_{i=1}^{\beta}$ and their corresponding forecasts (the $\alpha + 1$ ith) to compute an indicator of the prediction error. This indicator is compared to the threshold η, when violated, a deviation alarm of degree γ (default $\gamma = 1$) is raised. γ is sent to the node for updating (decreasing) α then T_α.

For computing the prediction error indicator, we have investigated two main metrics used in time series forecasting:

1. **Mean Absolute Percentage Error (MAPE)**:

   It is a measure of the accuracy of forecasting methods. It's expressed as a percentage and is computed using Eq. 3.1.

$$MAPE = \frac{100\%}{\beta} \sum_{k=1}^{\beta} (|\frac{\hat{s_k} - pred(\hat{s_k})}{\hat{s_k}}|) \tag{3.1}$$

   MAPE is simple to compute and easily interpretable but it presents some issues: division by zero, for too low predictions the percentage error can exceed 100%, negative errors ($\hat{s_k} < pred(\hat{s_k})$) has heavier penalty than positive errors, etc. Some enhanced versions were proposed to overcome these issues like *sMAPE* (Symmetric Mean Absolute Percentage Error) *MAAPE* (Mean Arctangent Absolute Percentage Error).

2. **Mean Absolute Scaled Error (MASE)**:

   MASE is also a measure of the accuracy of forecasts proposed in [Hyndman 2006]. It has many desirable properties compared to existing forecasting error measurements (e.g., MAPE). MASE is computed using β last forecasts, β last collected data-points, according to the training set (previous data-points) used by the forecasting model (See Eq. 3.2).

$$MASE = \frac{1}{\beta} \sum_{k=1}^{\beta} (\frac{|\hat{s_k} - pred(\hat{s_k})|}{1/(\beta - 1) \sum_{k=2}^{\beta} |\hat{s_k} - \hat{s_{k-1}}|}) \tag{3.2}$$

   It compares the actual forecasting method to the one-step naive forecast computed in-sample with a ratio of their respective Mean Absolute Errors (MAE). Then when MASE < 1, the actual forecasting performs well than the naive one and vice versa. We transform this threshold of 1 to η.

Concerning uncollected data-points forecasting, the two following statistical techniques are investigated:

1. The popular Box-Jenkins ARIMA family of methods shortened as **ARIMA (AutoRegressive Integrated Moving Average)**:

ARIMA exploits information embedded in the autocorrelation pattern of the data. Estimations are done based on the maximum likelihood. The forecast is done, on previously collected data points, using the univariate statistical time-series ARIMA model. ARIMA parameters (p, d, q) are computed automatically using Auto ARIMA[2]. It chooses the best model order, selecting the combination that minimizes model quality estimators such as the Akaike Information Criterion (AIC) or the Bayesian information criterion (BIC). For convenient time series forecasting, we need to satisfy a minimum training set size of 50. Then, we define a specific β, β_0 equals to 50 (default) for α equals to 0 in the beginning.

2. **Holt's Trend Corrected Exponential Smoothing**:

   Also called Double Exponential Smoothing, it is used to do forecasting when the time series is increasing or decreasing, meaning it presents a trend without a seasonal pattern. For this model, we may retain two main hyper-parameters: α^* the smoothing factor for the level (mean) and β^* the smoothing factor for the trend or slope smoothing.

The parameters used in our overhead reduction proposition are summarized in TABLE 6.1.

Table 3.1: Parameters

Parameters	**Description**
α	Confidence index on the ability of the prediction method to provide good estimations of uncollected data-points., $\in$ [0,..,9], $\alpha \in \mathbb{N}$
T_0	The required time granularity for statistic freshness (updates)
$T_\alpha = (1+\alpha)T_0$	Effective pushing period for the confidence index α
β_0	The minimal number of samples to be collected at the beginning on T_0 that will enable good forecasting enabling the adaptive push
β	Number of samples for a given α after which a deviation checking is done allowing α to increase or decrease
η	A threshold on the prediction error that triggers deviation alarms for decreasing α
γ	The degree of a deviation alarm

Let's recapitulate with the high-level workflow of our adaptive continuous monitoring scheme as shown in Figure 3.3. Initially, a certain application, control, or management service expresses its interest in certain information (1a). It specifies the metric M to be collected, at which frequency it may be available with T_0, and η

[2]http://www.alkaline-ml.com/pmdarima/develop/index.html

the collection (or forecast) imprecision tolerance indicator. The registrator then initialize the collection process on the node (1b) and (1c). The adaptive scheme then consists of adjusting the collection period for a push-based monitoring model according to α. Indeed network devices stream to the SDN controller each T_α (equals to $T_{\alpha min} = T_0$, initially) unit of time and this reporting interval increases automatically $[T_{\alpha min}, T_{\alpha max}]$ over time (2a). The raw collected data points are augmented with predictions and delivered to applications (2b). The reporting interval will be adjusted (decreased) only if the receiver side cannot capture the runtime evolution of the information being collected and thus can not guarantee a certain accuracy, initially defined. Then, it raises a deviation alarm, and the reporting interval is decreased (3).

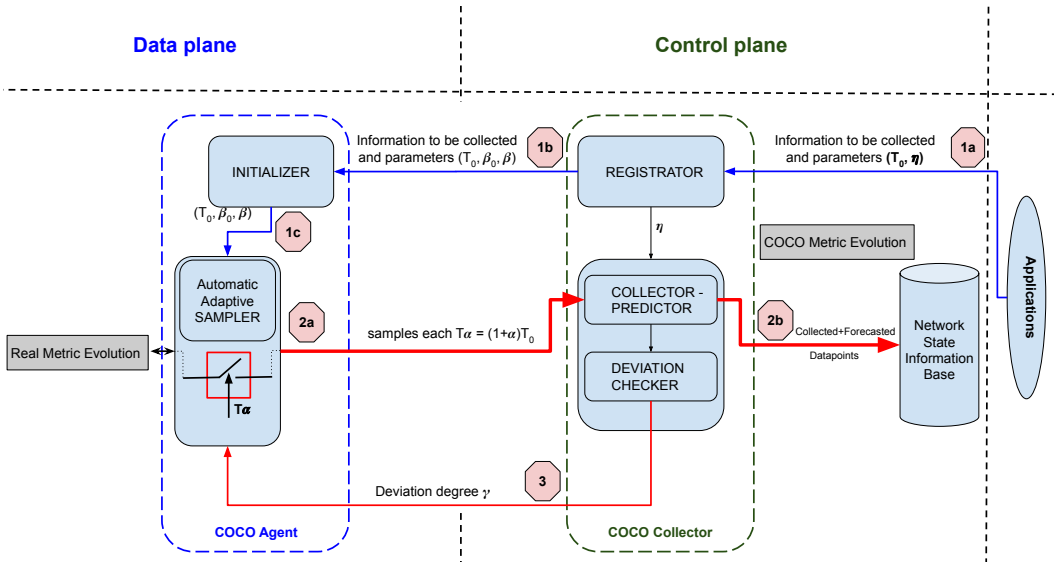

Figure 3.3: COCO High Level Workflow

3.3.3 Implementation

In this section we give some details on how the architecture presented in Figure 3.2 is implemented. The COCO Agent is implemented as a python script that could be run on any OpenFlow switch (e.g., OpenVSwhicth supported). At pushing point, locally, we use the command-line tool *ovs-ofctl*[3] to retrieve the flow statistics before sending them to the collector. The messages between the collector and the agent are carried with UDP Datagram sockets. The COCO Collector is built upon Ryu[4], a python-based SDN controller providing well-defined API to develop SDN control and management applications. For ARIMA forecasting, we use *pmdarima*, which is a Python and Cython wrapper of several different statistical and machine learning libraries (statsmodels and scikit-learn) and operates by generalizing all ARIMA

[3]http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.txt
[4]https://osrg.github.io/ryu/

models into a single class. For Holt's Trend Corrected Exponential Smoothing forecasting, we use statsmodels.tsa that contains models, classes, and functions for time series analysis.

3.4 Performance Evaluations

3.4.1 Setup and Evaluation Metrics

We evaluate our proposition using the topology in Figure 3.4 that was set up on a virtual machine of 32 GB of RAM and 20 vCPUs INTEL server powered by two very high-end 18-core Intel Xeon E5 2699 v3 CPUs and 64GB of RAM.

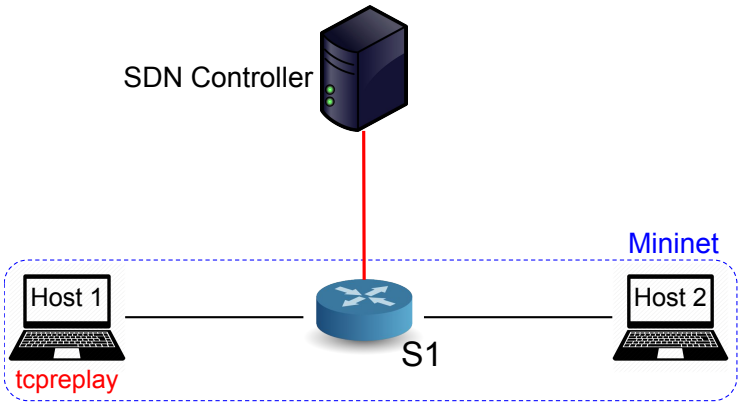

Figure 3.4: Experimental Setup Topology

With tcpreplay[5], we replay real network traffics from the two public anonymized datasets described below:

- **UNIV2 [Benson 2010]**: A university data-center trace collected on 22 January 2010. We extract 8 samples of 450s from 19:02:15 to 20:10:00.
- **MAWI [Sony 2000]**: A backbone trace from the WIDE MAWI archive collected in September 2019 at the transit link (1Gps) of WIDE to the upstream ISP. We extract 7 samples of 450s from the archives from 19 to 22 September.

Host1 executes tcpreplay that replays traces on the OpenFlow switch S1, which executes the COCO Agent. All the traffic from H1 follows a flow-rule and is forwarded on Host2, which is here just like a sink node. Host1, Host2, and the switch S1 are emulated with Mininet[6]. The SDN controller (Ryu) executes the COCO Collector and collects the flow statistics on S1.

Two main metrics are used in this evaluation. We compute these metrics compared to a fixed push mechanism at T_0 (which will be the case when we use, for example, sFlow for flow statistics collection).

[5]https://tcpreplay.appneta.com/
[6]http://mininet.org/

Accuracy: To evaluate the collection error induced by our adaptive push mechanism compared to a fixed T_0 one, we use the MAPE. It is expressed in percentage; the lower this value, the better it is.

Overhead: We express the overhead reduction with the percentage of the number of messages effectively pushed by the switch to the controller in the adaptive mechanism compared to the fixed one. We want this value to be as high as possible.

3.4.2 Overhead Reduction and Collection Accuracy

Figure 3.5 and Figure 3.6 show the percentage of messages reduced (on the left) and the collection error with MAPE (on the right) according to the threshold η respectively for the traces UNIV2 and MAWI, for (β_0, β) equals (50, 10) and (100, 10). We experiment using both ARIMA and Double Exponential Smoothing (ES) for uncollected data-points prediction, and we use MASE as the error indicator metric.

For all $\eta > 0$, meaning a fraction of inaccuracy is authorized on the collection, we always have a reduction of the overhead compared to the fixed periodic push collection. This reduction increases when η increases and may stabilize. We achieve up to 75% of reduction. And we have this if a very low error (less than 0.5 %) on the whole collection compared to the fixed T_0 collection. This very low error for both MAWI and UNIV2 increases with η and the reduction percentage.

Figure 3.7, shows for the trace UNIV2, the evolution of the overhead reduction percentage and the collection error for different values of the collection duration considering 4 combinations of (β_0, β): (50, 10), (50, 30), (100, 10), (100, 30). We can observe a slight increase in the reduction percentage when the collection duration increases, with always no collection accuracy degradation (less than 0.25%). In other words, the collection duration doesn't have a great impact on the overhead reduction or collection accuracy, which is normal since we are not looking for long-term forecasting where long history would be of great importance. Instead, we are making closed future predictions by forecasting near future data points. In this configuration, we do not need to go further in the past to have good forecasts. In other words, we emphasize recent data points supposing older ones less relevant, and because we are making an online prediction, we set a sliding window of size W = 1.5*β_0.

3.4.3 Computation Time and Evaluations Summary

When a data-point $\hat{s_i}$ is collected, before collecting the next one ($\hat{s_{i+1}}$) some computations are done, like computing estimations of missing data-points, and sometimes deviation checking. This inter-data-points processing time needs to be bounded for our algorithm's good functioning. This compute time may also impact the possible values of T_0. In Figure 3.8, we show the range of this computation time for both ARIMA and Exponential Smoothing forecasting. Exponential Smoothing gives better results (around 0.025 seconds). ARIMA gives computation time around 0.125

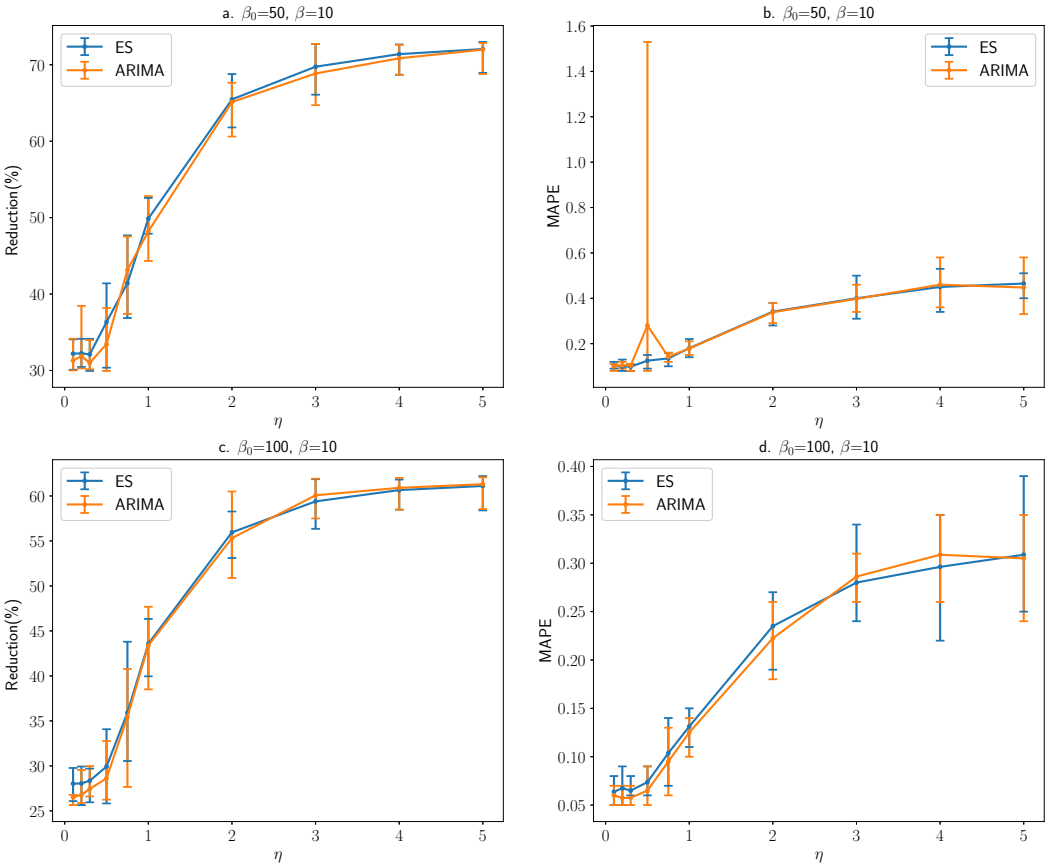

Figure 3.5: UNIV2: Reduction compared to a fixed T_0 pushed mechanism in % vs Collection error (MAPE) in %

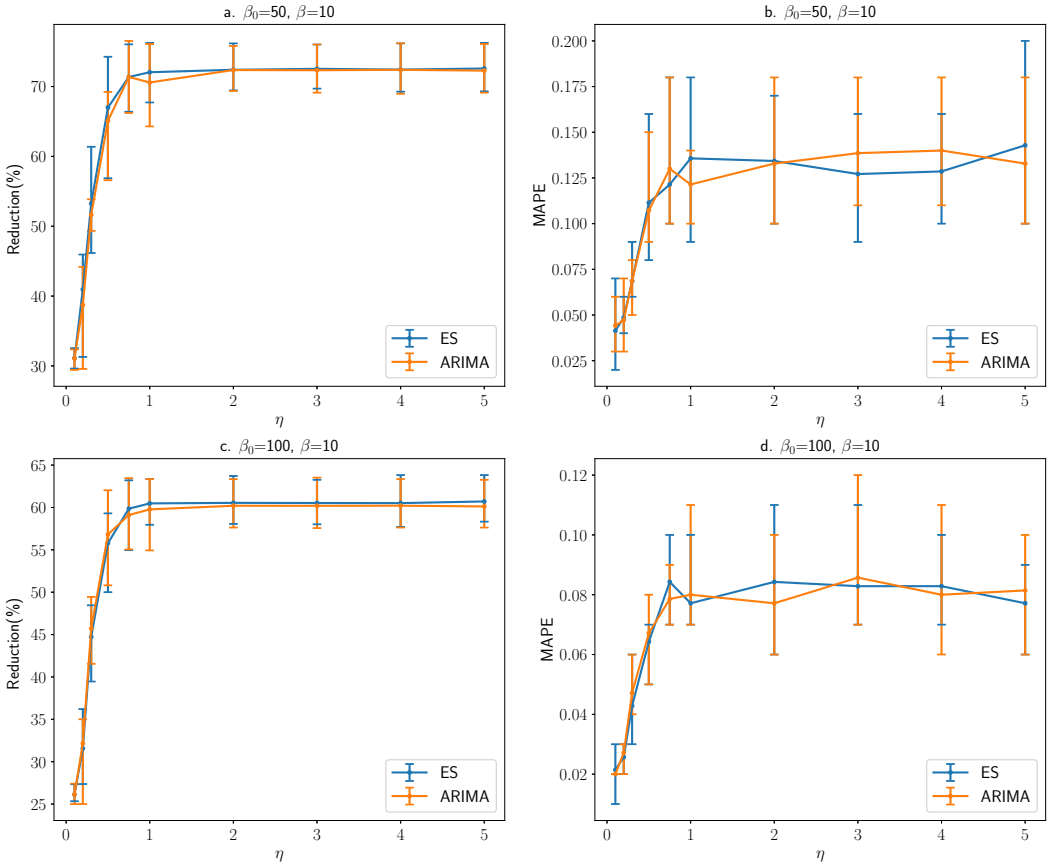

Figure 3.6: MAWI: Reduction compared to a fixed T_0 pushed mechanism in % vs Collection error (MAPE) in %

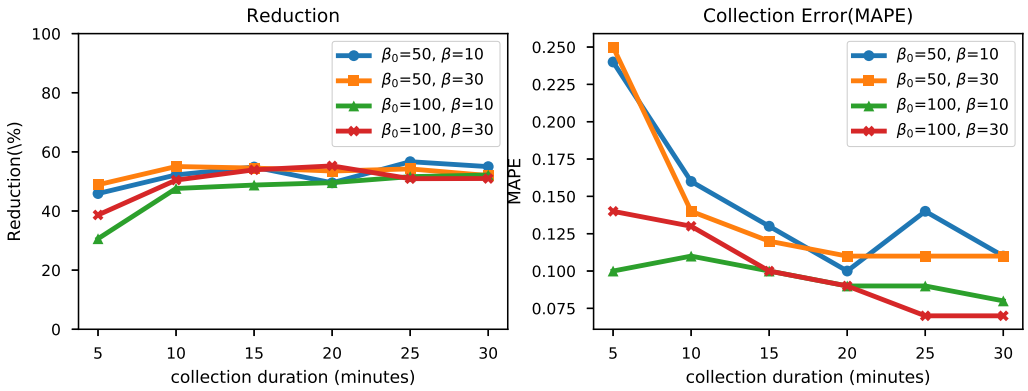

Figure 3.7: Reduction and Collection Error (MAPE) according to collection duration for $\eta = 1$

seconds. This value is high because in our implementation with ARIMA, when we receive the first data point at a new *alpha*, we rebuilt the forecasting model, which consumes a big amount of time resources (e.g., 0.5s or 1s even 2s). For the other cases, we just refresh the model with the newly collected data-point, and in these cases, the computation value varies around 0.03 s like Exponential Smoothing.

From Figure 3.5 and Figure 3.6, ARIMA and ES provide the same quality of forecasting even with the big difference in terms of computation time. Hence for our the scenario considered in this work, the better choice for uncollected data points is Double Exponential Smoothing. The scenario concerns time series with an increasing trend without seasonality pattern representing an aggregation of several flows, all the traffic collected on a backbone link (MAWI), or a selected switch in a data-center (UNIV2).

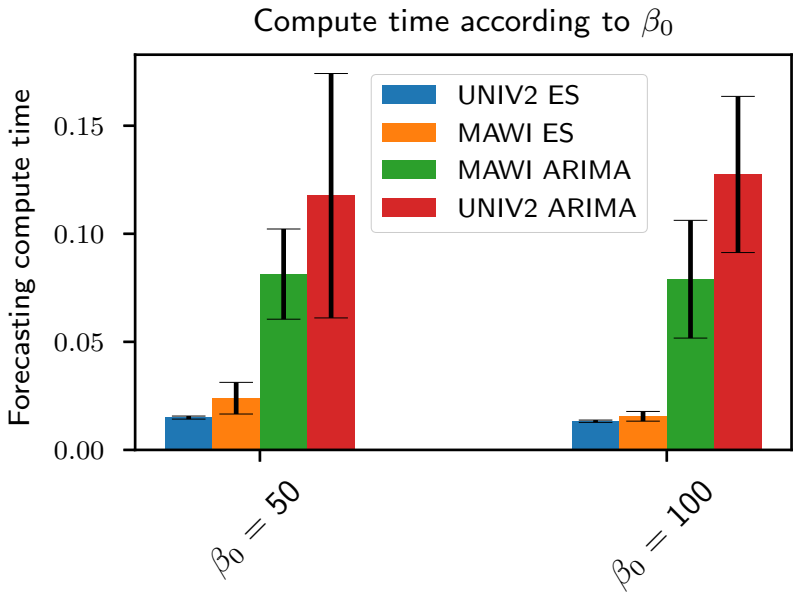

Figure 3.8: Compute time in seconds

3.4.4 Use Case: Network utilization Monitoring (Bandwidth Estimation)

Network utilization representing the percentage of a network's bandwidth currently being consumed by network flows or traffic, is an essential metric in operating a network and for cost-saving operations. The actual utilization of network resources (links, ports, switches) is difficult to measure, predict. As mentioned in [Hassidim 2013], looking at summary statistics will not tell the whole story since the utilization changes quickly, depends on many parameters (traffic demands, routing scheme, etc.). Using this view may not lead to good conclusions. We need to monitor performance metrics, such as network utilization continuously, to quickly

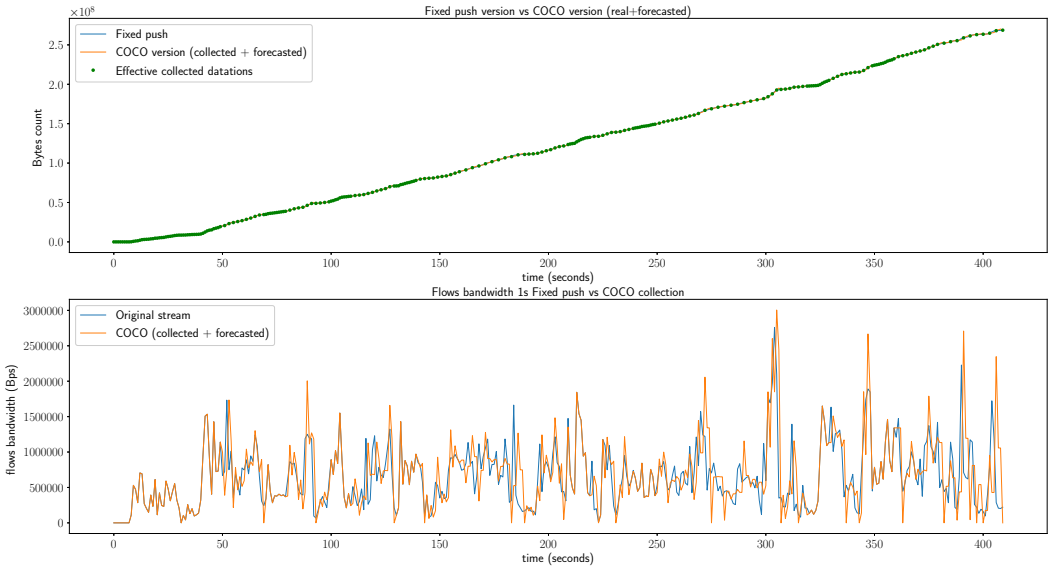

Figure 3.9: Bandwidth Estimation with COCO compared to a fixed push mechanism

adapt forwarding rules in response to changes in data-plane workload [Yu 2013]. Provide this continuous monitoring service but with low overhead is what we have done in this work.

With our lightweight periodic flow statistics data collection, we contribute to Traffic Engineering (T.E) in SDN by enabling the monitoring and analyses of near real-time network traffic information. T.E can then find reasonable routing mechanisms to guide network traffic, improve utilization of network resources, and better meet requirements of the network QoS, energy-saving scheduling. **Bandwidth management** that consists of measuring and controlling the communications (flows, packets, traffic) on a network link – to avoid overfilling – is an essential element of T.E. Indeed, the overfilled link may result in network congestion, and poor performance of the network[7].

Figure 3.9 shows bytes count evolution collected with our adaptive push mechanism compared to a fixed push collection and the bandwidth estimated from this statistic. The bandwidth is expressed in bytes per second (Bps).

3.5 Discussion: Scalability and INT/P4

The objective here is to position our work with scalability concerns and the recent directions in Software Defined Networking intending to take complete control over the data-plane with the advent of P4, INT, and programmable switching.

[7]https://en.wikipedia.org/wiki/Bandwidth_management

3.5.1 Scalability

This discussion is motivated by this question that may arise: how can this push and prediction-based adaptive collection scales or works with a typical SDN flow-based network where the data-plane is composed of many switching elements? There is no need to worry about this aspect. Indeed, many works have handled the problem of SDN monitoring overhead reduction by proposing to select a few numbers of switches, called key switches, that may cover all target flows or links of interest under monitoring. The key switches are sometimes simply the ingress and egress switches of flows. More efficiently, their selection is formulated as a weighted set cover problem [Su 2014] or a vertex cover problem [Bonfoh 2018]. This key switch selection allows can reduce monitoring cost without loss of accuracy.

Our overhead reduction is to be done on these selected switches, using adaptive push instead of polling used in the solutions mentioned above. This can then reduce the overhead imposed by the collection at high frequency to ensure timely information. Our solution may be seen as time dimension optimization versus space dimension optimization of the key switches selection solution. And finally, combining these two approaches may reduce the overall monitoring cost for constructing the global and up-to-date view of the state of the data-plane in SDN.

3.5.2 P4/INT

One of the objectives of the couple SDN/OpenFlow was to give operators control over their networks instead of being dependent on the equipment vendors. And this has been accomplished in some ways with good control plane abstractions and the "match-action" forwarding model (e.g., big data-centers like those of Google built their homegrown control planed based on open interfaces exposed by the data-plane). But the control will not be effective if network owners are not in charge of how packets are processed by network elements [Casado 2019]. This leads to the advent of programmable forwarding chips (PISA - Protocol Independent Switch Architecture) and P4 (Programming Protocol Independent Packet Processor) [Bosshart 2014], a high-level language to express how a switch is to be configured and it has to process packets. This will enable removing the obstacle of fixed functions switches that recognize a predefined set of header fields and process packets with a small set of predefined actions.

It's worth pointing out that P4's goal is not to replace OpenFlow, but OpenFlow may be seen as part of P4 and is one of many possible programs (e.g., P4Runtine) to describe the forwarding behavior. Since there, our OpenFlow-based implementation can be considered P4-compatible.

Generally implemented with P4, INT (In-band Network Telemetry) [Kim 2015] is a new abstraction that allows data packets to collect switch internal state (queues size, queues latency, byte counts, etc.), enabling monitoring of the network state by the data-plane. We sometimes need to collect from a selected node statistics to a central server even with this in-band approach. For example, in [Harrison 2018]

network-wide heavy hitter detection is done with low overall control and data-plane communications overhead. The large flows are detected locally in the data-plane with a per-key threshold that triggers reports to the control plane (central coordinator). A similar approach is used in [Castanheira 2019]. At the triggering moment when a sort of near real-time statistics collection by the control plane may be done, our adaptive solution will reduce the overall monitoring task cost.

3.6 Related Work

For continuous statistics collection in SDN, especially flow bytes count, traditional network monitoring flow sampling techniques like sFlow, NetFlow/IPFIX could be used. But we don't need this extra instrumentation on OpenFlow Switches since they already provide flow statistics by maintaining records on packets matching installed flow rules. We just need to collect these statistics efficiently. However, OpenFlow will face some issues to provide the same level of flow-level measurements compared to the sampling techniques mentioned above, especially the limitation of the number of flow rules that a switch can support. In light of this, [Suárez-Varela 2017] emulates NetFlow/IPFIX operation in SDN, providing OpenFlow compatible flow sampling methods. For near-real-time statistics collection with good accuracy-efficiency trade-off, [Cheng 2017] proposes per-flow sampling with adaptive polling frequency. The polling frequency is adjusted depending on whether the sampled traffic is stable or busy. In the same context, to have a good accuracy/overhead trade-off while providing timely data-plane state information, [Chowdhury 2014] proposes a threshold-based adaptive scheduling algorithm for flow statistics retrieving by polling

FlowSense [Yu 2013] proposes for continuous link utilization monitoring in SDN, a zero-overhead passive technique using PacketIn (arrival of a new flow) and FlowRemoved (expiration of a flow entry) messages to estimate the link utilization. This solution was qualified as a push-based approach, but we consider it more like an event notification one (see section 3.1). FlowSense has the advantage, with its passive measurement, of not imposing any additional monitoring overheads. Still, if there are long flows, it doesn't give good estimations since we have few flow expiration events.

Lots of adaptive algorithms are provided in the literature from wireless sensor networks data collection [Laiymani 2013] and Internet of Things (IoT [Trihinas 2018] to SDN for monitoring overhead optimization. As pointed out in [Tangari 2018], we can distinguish two main techniques:

- **Threshold-based adaptive monitoring**, where the monitoring period is adjusted by comparing the two last measurements variations to upper and lower thresholds.

  Payless [Chowdhury 2014] uses this approach. Its authors focus on the trade-off between monitoring accuracy and network overhead by proposing a frequency adaptive statistics collection scheduling algorithm applied on selected

switches. Payless's adaptive algorithm adjusts polling frequency dynamically based on collected data granularity.

- **Prediction-based adaptive monitoring**, where the next measurement is predicted with the last collected ones. The deviation between the forecast and the real observation guides the period adjustment.

  This approach is used in [Zhang 2013] which uses a linear prediction-based dynamic adjustment scheme to provide dynamic zooming into the flow space (temporal and spatial dimensions).

Authors in [Tangari 2018] propose SAM (Self-tuning Adaptive Monitoring) which uses a prediction-based approach to adjust polling rate dynamically but with minimal parameter tuning effort, not like [Zhang 2013] and [Chowdhury 2014].

Other relevant works on the topic include [Van Adrichem 2014] and [Fawcett 2018]. OpenNetMon [Van Adrichem 2014], for traffic monitoring (flow-related information collection with OpenFlow), uses an adaptive polling rate that increases and decreases respectively when the measurement values differ between samples and when the values stabilize. TENNISON [Fawcett 2018] is a distributed security framework with effective and proportionate monitoring. It offers multi-level monitoring using appropriate tools (sFlow, IPFIX, DPI) from layer 1 to layer 2. Its sampling and polling rate are dynamically adjustable based on three thresholds.

Although the adaptive monitoring works presented above are very interesting, they are mainly tailored for traffic monitoring using a polling approach on the first hand. And on the other hand, most of them require complex parameters tuning. We do not need only flow-related information collection with a poll-based approach for fine-grained management at the SDN controller level and within the self-driving network's vision with its real-time telemetry requirement. But all relevant information that may contribute to the decision-making process may be collected efficiently as proposed on Figure 3.1.

We propose COCO, an adaptive data collection framework for a push-based monitoring model with low parameter tuning for the special case of continuous statistic collection. It uses the two main techniques presented above: a **prediction technique** to forecast uncollected data-points and a **threshold decision making process** for deviation alarm raising, that expresses if T_α should be increased or decreased (section 3.2.3). It also has the advantage of being lightweight for source nodes.

3.7 Summary

This chapter presents an SDN monitoring framework clarifying information that needs to be collected in an ad-hoc manner with classical OpenFlow polling techniques, events that need to be conveyed by notifications, and periodic statistics collection for which we propose a trivial push model. This proposition may allow

reducing the periodic monitoring cost (as the number of messages through the control network) by 50%. But to go further, we propose an efficient adaptive push algorithm for the near-real-time flow statistics collection. It is based on time series prediction (ARIMA and the Double Exponential Smoothing) that guides the adjustment of the pushing time. Our proposition has the advantage of being lightweight and easy to deploy. We evaluate it using real-world backbone and university datacenter traces. We reduce the collection overhead up to 75% compared to a fixed periodic pushing collection mechanism at the price of almost no accuracy degradation with less than 0.5% of collection error.

Part II

ML-based SD-DCN Management

Chapter 4

SD-DCN and AI/ML Preliminaries

In the last few years, data centers have grown considerably to accommodate the ever-growing Internet services and applications. Enterprises and corporations are gradually moving their services like Web search and online gaming into cloud environments. The cloud itself is a collection of data centers. DCs are not only the backbone of cloud computing but also the vertebral column of the Internet. Hence, it is a complex and dynamic environment with complex problems where traditional management struggles to provide good performance. It was naturally managed with centralized traffic engineering as a controlled environment, making it one of the first network environments where SDN was deployed. In this second part, we leverage AI/ML on top of SDN to cope with these complexities by assisting the human operators or removing them from the control loops for DCN management tasks

This chapter aims to provide background information on data center networking and the problem we consider. Firstly, Section 4.1 provides an overview of data centers and how they are organized. We end this section by presenting the incast and elephant traffic management challenges. Then, Section 4.2 provides AI/ML definitions and the main ML algorithms categories. After that, the rest of the section presents a holistic review of previous works that use ML to address network problems in general and address problems in the SDN context. We conclude this chapter with Section 4.3 by proposing an AI/ML approach for incast and elephant traffic management as ML-based management in Software-Defined Data Center Network

(SD-DCN). We took insights from the already application of ML to networking and the lessons learned from these previous studies.

4.1 Data Centers

4.1.1 Data Centers 101

A data center is a physical facility that organizations use to house their critical applications and data. A data center's design is based on a network of computing and storage resources that enable the delivery of shared applications and data. The key components of a data center design include routers, switches, firewalls, storage systems, servers, and application-delivery controllers[1]. Information and resources (hardware and software) from these data centers available to users over the Internet is denoted as cloud computing.

Cloud computing, unlocked by virtualization, brings more outstanding capabilities and opportunities for recent IT resources (hardware (CPU, RAM, disk, GPU, I/O, etc.) and software) utilization. With data centers, they have become the de facto hosting platform for all social and economic innovations. For example, any IoT deployment will likely have a cloud in the backend. We can say the same for other types of deployments, such as smart mobility, big data, industry 4.0, and public cloud AI. More and more companies migrate business from their own servers to the cloud.

The cloud is then present in several aspects of our life. Besides traditional Web services such as Web mail, searching, and online education, billions of devices in the Internet of Things (IoT) also rely on the cloud to host their data [Xu 2017]. The data volume is undergoing extremely rapid growth. Massive amounts of data are generated in DCs. Many data centers have been constructed worldwide due to the explosive growth of data volume and type.

In recent years, many data centers have then been built to accommodate this explosive growth of data. Many companies, such as Amazon, Microsoft, Google, Facebook, have spent billions of dollars establishing these data centers. Huge volumes of hardware, software, and database resources in these large-scale data centers can be allocated dynamically to millions of Internet users simultaneously [Zhang 2016].

A map of data centers in the world (4777 colocation data centers from 127 countries) is presented in Figure 4.1[2].

Thousands of millions of machines from these data centers (e.g., 50k-80k servers for a single typical Amazon DC) serve the Internet's 5 billion users via Amazon, Google, YouTube, Microsoft, Facebook, Netflix, Twitter, only to mention this. For example, Facebook data centers serve 2.6 billion active users daily. End-users interact with these powerful computers through the cloud. Data centers deliver seamless

[1] https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html?dtid=osscdc000283

[2] https://www.datacentermap.com/

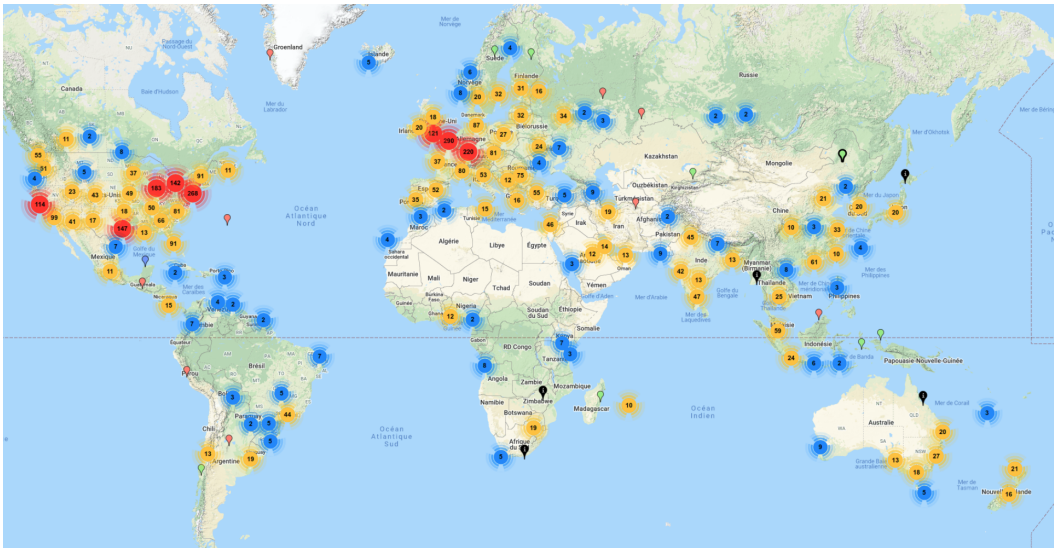

Figure 4.1: Data centers World Map

storage and fine-grained distributed computing, especially for high computing requests from AI/ML applications. DCs are not only the backbone of cloud computing but also the vertebral column of the Internet. DCs keep the Internet going even when events like COVID-19 trigger historic surges in traffic[3].

4.1.2 How to make the Data Centers work ?

As a result of the section above, making internet works pass through making the data centers work. Indeed, data centers have evolved from private and small-sized to large-scale warehouse facilities to accommodate the ever-growing Internet services and applications. A data center facility typically houses a large number of computing and storage nodes interconnected by a specialized large-scale networked system in a centralized and controlled environment called a data center network (DCN) [Xia 2016]. Therefore, making the DC work means designing and operating the data center networks (DCNs) efficiently to meet traffic demands, heterogeneous performance needs, and scale. These requirements pose great challenges to DCN infrastructure and its management.

On the first hand, concerning the DCN infrastructure establishment, specialized technologies and hardware were designed. For example, Amazon uses custom routers/switches based on Broadcom Tomahawk ASIC silicon (very high bandwidth Ethernet Switch Chip at 25.6 Terabits per second). New transmission technologies (optical fiber-based) were also designed. New typologies to handle the complex data center environment are studied, tested, and deployed in the same context. The most famous one was the 3-Tiers, and finally, the Leaf-Spine fabrics are dom-

[3]https://www.google.com/intl/fr/about/datacenters/podcast/

inating (See Figure 4.2)[4]. For the high dynamicity of DCNs, continuous hardware software/operating systems upgrading is needed.

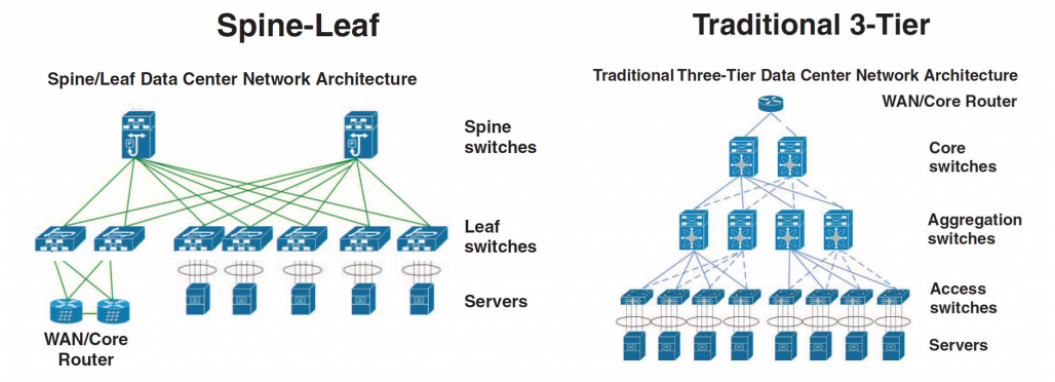

Figure 4.2: Leaf-spine vs 3-tier architectures

On the other hand, DCN management operations include flow scheduling [Alizadeh 2013, Chen 2016a, Chen 2018], congestion control schemes [Alizadeh 2010a, Kumar 2020] , load balancing [Alizadeh 2014, Katta 2016, Zhang 2018], switch buffer management [cis 2017], etc. All these management tasks' main objective is traffic performance optimization with efficient resource utilization. energy consumption is also one of the big optimization concern.

It's was clear that with the complexity of data center networks and their traffic, these management operations need to be automated as much as possible. More, as a controlled environment, a data center was naturally disposed of for a centralized management approach [Zhang 2018]. SDN was then rapidly adopted for DCN management. Indeed Google was one of the first pioneers of SDN adoption [Clark 2016]. Google employed SDN principles to build Jupiter [Singh 2015], a data center interconnect capable of supporting more than 100,000 servers. As of 2013, it supports more than 1 Pb/s of total bandwidth to host its services. Furthermore, Google built an SDN-based network B4 [Jain 2013], a private backbone network connecting its data centers across the world. With this SDN management, Google operated its continuous evolving networking with big challenges that traditional and entirely distributed methods were unlikely to manage.

With centralized network architecture such as SDN, some management operations can adopt a logical-centralized controller to collect global network information and efficiently operate the underlying network resources leveraging the global view of the data center network. Operations automation is also eased. SDN automates the configuration of individual network elements from the logically centralized controller. SDN helps make better decisions more rapidly than would not be possible with a fully decentralized mechanism.

Since the control algorithms are running on dedicated servers in the control plane, the data plane elements, especially switch operations, are more focused and

[4]https://community.fs.com/blog/connecting-cisco-nexus-9396px-40g-spine-leaf-netowrk.html

specialized for forwarding, their primary function. More, innovations can then be made independently in these two planes, without relying on any external vendor, which is a good deal. The SDN architecture, this way, increases the overall performance.

Despite its advantages, SDN is not necessarily more straightforward than the existing architectures. Still, it offers some distinct benefits in enabling rapid evolution, greater specialization, and increased efficiency. Among the challenges of the SDN architectures, the control software, meaning the management operations algorithms, needs to be developed efficiently.

The increased autonomy with SDN is good but not sufficient regarding classes of traffics with heterogeneous performance requirements present in DCs and many optimization opportunities left. The research of more and more autonomy in management tasks must then be conducted, and AI/ML can help. We will see Why and How in the next Section 4.3, especially for incast and elephant Traffic Management in SD-DCN.

4.1.3 Incast and Elephant Traffic Management Problem in DCN

Modern cloud-native applications (big data analytics, IP-storage, etc.) brings new traffic patterns on data center networks (DCNs) [Wang 2018, Alipourfard 2017]. The DCN workload is thus composed of more and more server-to-server traffic that include essentially long-lived flows or elephant flows (e.g. backup, replication, data mining, etc.) and short flows or mice flows (e.g. delivering search results). Besides this classification, datacenter workloads often require sending requests to large numbers of servers and then handling their near-simultaneous responses, causing a problem called *incast* [Handley 2017]. This many-to-one communication and its associated traffic pattern are also designed as *incast traffic*.

This many-to-one pattern in data centers is used for applications such as distributed storage (e.g. BigTable, HDFS and GFS), web-search with partition/aggregation design pattern, and cluster computing platforms (MapReduce, Spark, etc.) [Zhang 2012]. Depending on the size of the responses of the servers we can distinguish between long-lived incast and short-lived incast. But it's worth mentioning that incast manifests generally in the short-lived form [Phanishayee 2008].

The coexistence of incast and elephant traffics with heterogeneous QoS requirements (high throughput for elephant traffic and low completion time for incast) complexify data center management tasks. These tasks include, naturally, congestion management, which aims to mitigate congestion, one of the principal causes of performance degradation. We can distinguish congestion control [Alizadeh 2010a, Kumar 2020, Hu 2020] and buffer management (buffer sizing and AQM - Active Queue Management) [cis 2017, Gomez 2019, Chuprikov 2020]. Examples of congestion control algorithms are Cubic, NewReno, DCTCP, BBR, etc., and AQM scheme examples are RED, CoDel, Fq-CoDel, etc. The other management tasks concern routing, load balancing, etc.

Indeed, managing incast traffic patterns with low latency using classical trans-

port protocols without switch assistance is very challenging [Handley 2017]. The network's response to uncontrollable incast traffic is to drop its packets by tail-drop queues [Xu 2019]. These packet drops degrade considerably in incast performance and the co-existing traffic, including long-lived flows or elephant traffics [Alizadeh 2013]. Consequently, the network requires smart packet buffering capabilities to efficiently handle the performance needs of these data center applications [Xu 2019, cis 2017].

In this part II of the thesis, we then focus on **smart and adaptive buffer management**, a switch-level operation, to provide great performance for both incast and elephant traffics in dynamic DCN environment. The smart buffer management may consist of an autonomous performance optimization process, that involves continuous network monitoring (performance and resource utilization) and performance modeling[5]. On the first hand, continuous monitoring is needed to have deep visibility on the network. It may ease autonomous optimization through continuous adjustment. This can be done with SDN, with its telemetry capabilities and its management flexibility. On the other hand, modeling is fundamental for network optimization, as reported in [Rusek 2019]: "we can only optimize what we can model". Indeed, to optimize incast and elephant traffic performance, a model providing insights on how various factors affect them is required.

However, there is a lack of those of models. Indeed, the classical approach for network modeling is the design of handcrafted and specialized performance analytical models. For incast traffic, mostly carried with TCP, performance modeling is very challenging [Chen 2009]. Indeed, TCP's stack is a complex system that involves many heuristics to handle network conditions and application behaviors [Li 2019]. Subtle changes in its parameters may lead to completely different performance. The existing incast performance analytical models [Chen 2009, Zhang 2011, Chen 2015] are either tightly coupled with a particular protocol version or specific to certain empirical data. Relying only on analytical performance modeling for incast performance optimization is then not a practical solution.

Therefore providing optimal performance to incast and elephant traffics in dynamic DCN through smart and adaptive switch buffer management poses two main challenges:

- **Performance Modeling**. The need of a performance modeling approach that generalizes easily and that do not rely on any domain-specific assumptions or approximations.

- **Continuous Optimization**. The need for an efficient optimization process that could leverage the aforementioned performance model to anticipate and efficiently optimize buffer-management-related parameters to achieve optimal performance.

[5]https://blogs.cisco.com/cloud/how-to-strike-the-right-balance-between-application-performance-and-cost?ccid=cc001268

4.2 AI/ML for Networking

4.2.1 AI/ML 101

AI and ML have a slightly longer history than one might think at first glance. Indeed the term machine learning was coined in 1959 by Arthur Samuel, who defined it as a "Field of study that gives computers the ability to learn without being explicitly programmed.". To achieve this goal, ML leverages knowledge through experience and by the use of data. Typically, it builds a model based on sample data, known as "training data, " to make predictions or decisions without being explicitly programmed to do so.

ML is part of AI, a broader field that deals with building intelligent machines capable of performing tasks that normally require natural intelligence displayed by humans. ML can be seen more as a technique, and it is used to create AI. To go further and have intuitions about ML, it is legitimate to compare it with traditional programming. An illustration of this comparison is shown in Figure 4.3.

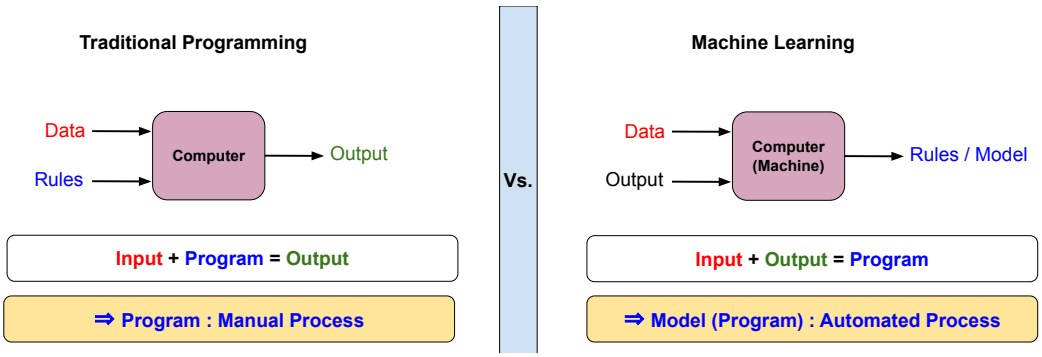

Figure 4.3: Traditional Programming vs Machine Learning

The manual process aspect of traditional software programming comes from designing programs or rules relying on step-by-step logic (if-else statements), loops (for/while), Maths/Boolean operators, etc., to solve a specific problem with a computer. These rules come from insights we have on the problem through experience. Unfortunately, this approach breaks down with some complex problems not necessary for humans but computers, such as image recognition. ML's alternative method consists of providing computers the ability to work like a human brain using examples (input and output) as a training dataset with a so-called trial and error mechanism. ML then learns from data. The resulted program is a predictive model that can be used for the prediction of future outcomes. AI/ML is being applied to solve complex problems in many fields, including image and speech recognition, robotics, finance, health care, business, etc.

There are three main types of Machine Learning algorithms: supervised learning, unsupervised learning, and reinforcement learning:

Supervised Learning (SL). It uses labeled datasets that contain both the inputs and the desired outputs or target to built prediction models. The resulting model

can then be used to predict the output associated with a new input. SL is used either for *regression* when predicting a continuous-valued attribute associated with an input (e.g., house prices) or *classification* when associating a category to the given output (e.g., email filtering as spam or non-spam). SL algorithms include Support Vector Machine (SVM), Decision Trees (DT), Random Forest (RF), Neural networks, and so on.

Unsupervised Learning (USL). USL, on the other hand, uses unlabeled data (only inputs) and tries to find structure in the data. This means clustering or grouping the data points. One of the central USL algorithms is K-means, which clusters data by separating samples in n groups of equal variance.

Reinforcement Learning (RL). RL consists of finding optimal control policies (actions to take by an agent) on a given environment typically represented with the Markov Decision Process (MDP) framework. By interacting with the environment, the agent learns optimal policies by trying to maximize a cumulative reward. RL algorithms include Q-learning, Qeep Q Network (DQN), etc.

4.2.2 AI/ML for Networking

After all, it's worth pointing out that the recent advancements in AI/ML technology have significantly benefited from and empowered by networking offering key infrastructure with efficient computational resources. In general, Ml benefits from modern specialized hardware and ML libraries developments. Optimized implementations and frameworks coupled with specialized hardware are designed. These hardware accelerators include FPGA (used, for example, by Microsoft and Xilinx ML Suites), Nvidia's GPU, AI ASICs (e.g., Google's TPU), etc. The ML libraries such as Scikit-Learn, Keras, Tensorflow, and Pytorch contribute to ML advancement and help to its democratization.

Networking also has to benefit from these recent AI/ML advancements. ML has then been leveraged by the networking community in academia and industry to deal with the complex problems faced within several research areas. ML brings the promise of more accurately handling complexities in communication systems where traditional analytic mathematical models struggle with the exponential growth of network traffic thanks to the advances in smart devices, the Internet of Things (IoT), and cloud computing. The traffic diversity with heterogeneous performance requirements complexifies a lot the underlying network design and management.

ML has already been applied in a wide range applications for networking [Boutaba 2018, Ridwan 2021] through several network types as WSN, MANET, cognitive radio networks. cloud, etc. The main applications ares include traffic engineering, performance optimization and network security. With traffic engineering, ML was applied to traffic prediction (e.g. [Edmund 1993, Chen 2016b, Li 2016]), traffic classification (e.g. [Kim 2008, Zhang 2014]) one of the earliest fields were ML was applied, and routing (e.g. [Hu 2010, Mao 2017]). ML application in performance optimization concern congestion control (e.g. [Geurts 2004, Winstein 2013]), QoS/QoE improvement (e.g. [Mushtaq 2012, Jiang 2016, Sun 2016]) and resource

management (e.g. [Mijumbi 2014, Mao 2016, Tayyaba 2020]). Finally when it comes to networking security one of the field when ML was leveraged a lot, we have intrusion detection system (e.g. [Mukkamala 2002, Alom 2015]) and anomaly detection (e.g. [Garcia-Teodoro 2009, Zhao 2015, Owezarski 2010, Mazel 2015]).

The typical ML application to networking workflow is very similar to the classical ML framework, as presented in Figure 4.4 specified in this work [Wang 2017].

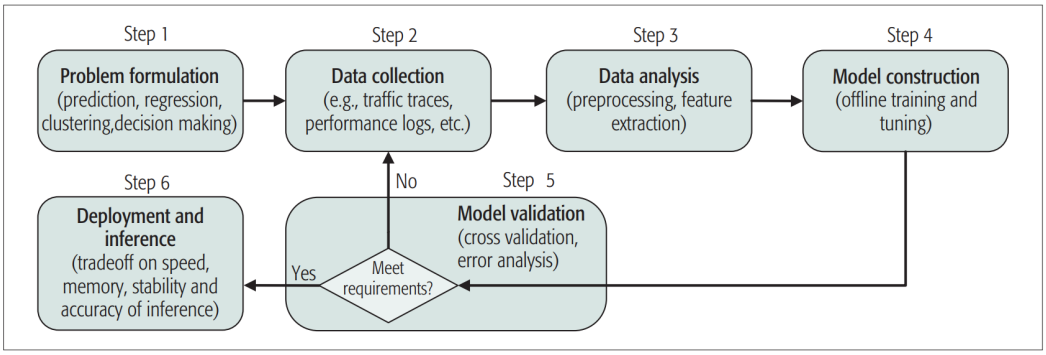

Figure 4.4: The typical ML Workflow for Networking (from [Wang 2017])

The plethora of ML applications to networking studies may give an idea about the exciting opportunities ML can bring to networking. However, with the traditional distributed nature of networks, the application of ML would not be at its full potential due to the lack of data collection capabilities and dynamic network operations and configurations. Thanks to SDN that can help with these concerns by easing ML application to networking management.

4.2.3 AI/ML in Softwarized Networks

With SDN's global visibility on the network and its programmability ML-based, optimal management operations can be executed on the network in near-real-time, even in real-time. Applying ML with SDN is then of significant interest to handle a wide range of network management tasks [Xie 2018, Zhao 2019].

These tasks are quite the same as in the previous section but in an SDN fashion and include resource management and allocation, network flow/traffic management, QoS/QoE prediction, routing optimization, and security. ML algorithms in SDN-based resource management and allocation (e.g. [Mao 2016, Martin 2018]) tend to maximize the utilization of the data plane resources, using mostly RL-based algorithms. ML in SDN-based traffic management concern traffic control and traffic classification (e.g. [Gao 2018, Xiao 2015, Amaral 2016, Wang 2016, Fan 2017]). For QoE/QoS predictions (e.g. [Jain 2016, Pasquini 2017]) leveraged either by proactive control operations or planning operations. Concerning routing optimization (e.g. [Pasca 2017, Azzouni 2017, Stampa 2017]) with ML and SDN, more efficient routing mechanisms proposed are traffic-aware and energy-aware. And finally, ML and SDN association provides interesting ways of securing the network

(e.g. [da Silva 2016, Shone 2018, Kalkan 2018, Sultana 2019]).

SDN relates to other technologies, especially Network Function Virtualization (NFV) that enables operators to virtualize functions like load balancing, routing, firewalling on commodity servers instead of using dedicated hardware. Both SDN and NFV come from a software-oriented networking paradigm to provide high flexibility in network operations, and fast but simple adaptation to network changes [Kellerer 2019]. The resulting networks are then referred as softwarized networks, but in this thesis, when talking about softwarized networks, the emphasis is made on SDN. As seen from the works presented above, these softwarized networks with ML as a data-driven approach for control and management are part of enabling technologies for future networks.

It comes out from the several ML applications to networking in general, especially in SDN's case, that even if there are great promises, open questions remain on the feasibility and practicality of ML approaches for modern and future networks management. These questions concern prediction cost, the need for adequate and realistic datasets availability, evaluation baselines, good exploration versus exploitation for RL-based algorithms, only to mention this. More, the ML application to networking exhibits disparate ML applications calling for a certain standardization or uniformization [Boutaba 2018] or even by being guided by a certain long-term vision as autonomic networking or self-driving networking (SelfDN).

4.3 Thesis Approach: AI/ML for Incast and Elephant Traffic Management in SD-DCN

With SD-DCN, the network is operated by software designed by operators in the control plane. Classical optimization methods that are there for a long are still used but more easily, in a proactive manner taking advantage of the centralization. However, decisions or critical decisions are still hardcoded based on insights gained by the specialists, their handcrafted analytical models with traditional optimization techniques. These optimization strategies concern multi-commodity flow problem [Zhang 2018], Integer Linear Programming (ILP) models [Huang 2018] while the problems we may want to solve with modern DCNs are very far from exhibiting linear patterns, etc. It's worth pointing out that even with this picture, things are still working, thanks to operators/managers' complexity mastering.

But with the magnification of the complexity, the approach described above can just not holds and may scale terribly in people costs [Clark 2016] but also in its ability to master this resulted complexity. And at this point, with the recent technological advancement in AI/ML, ML has a significant role to play by assisting management operations (humans implications, software) in complex, real-time decisions making in DCN dynamic environment. The researched goal is *to simplify management operations as much as possible* in SD-DCN with continuous evolving complexity. ML can bring modern automation and control through analytics and data-driven methods.

ML, powering historical data that in some sense is leveraged by human operators to master complexity, can provide performance models without the need of any domain specifics knowledge compared to the traditional approach using handcrafted and specialized analytical models. With the ML approach, the simplification aspect opens ways for more automation in the management operations.

Aligned with the SelfDN vision as introduced in Section 1.2, we proposed the base architecture on Figure A.1 for efficiently handling the mixed elephant and incast traffic. We will design certain building blocs in the two following chapters. In Chapter 5, an ML inference agent for predicting incast performance based on a variety of system parameters is presented. And in Chapter 6 we will propose traffic optimizations blocs for optimizing both incast and elephant performance leveraging the ML prediction agent developed.

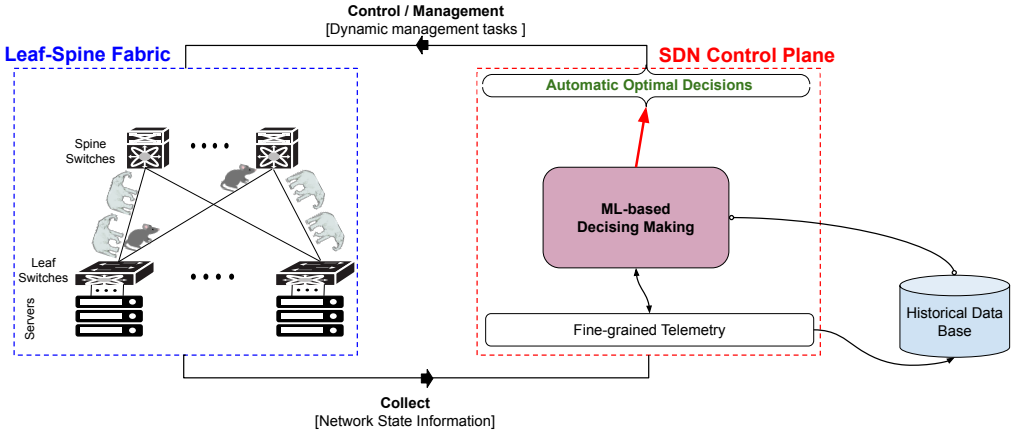

Figure 4.5: Towards SelfDN for SD-DCN Management

Figure 4.6 highlights the concerns and contributions of the second part of the thesis that uses AI/ML to brings more intelligent automation of complex DCN management tasks. The ML-based proposition takes advantage of SDN, prepared in the first part by augmenting its telemetry capabilities in terms of global visibility, programmability, and data collection.

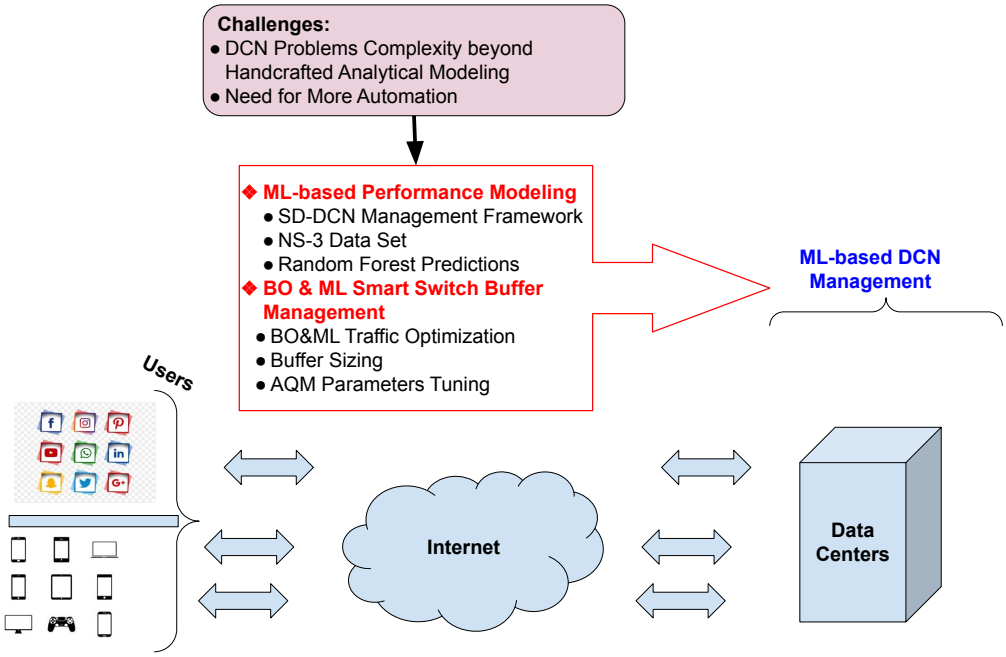

Figure 4.6: Thesis Part II: Intelligent ML-based Management in SD-DCN

Chapter 5

Learning-based Incast Performance Inference in SD-DCN

After presenting a high-level overview of DCN and how ML can help its management in the previous chapter, this chapter proposes a machine learning framework built upon SDN for incast performance prediction in DCN. This service can be leveraged by smart buffering schemes and online network optimization algorithms to provide efficient performance in data centers. Indeed, handling the critical incast traffic in DCN requires smart switch buffering. However, the smart buffer management process needs incast performance models that provide insights on how various factors affect it. Unfortunately, the literature lacks these types of models. The existing ones are analytical models, which are either tightly coupled with a particular protocol version or specific to certain empirical data. Motivated by this

observation, we propose the machine-learning-based incast performance inference approach using Random Forest.

The chapter is organized as follows. We first introduce the incast traffic and present the scenario setup used and the corresponding notations in Section 6.1. Then, in Section 5.2, we give a detailed presentation of our proposed framework. Section 5.3 presents the model construction stage. We carry out intensive experiments with the NS-3 simulator and construct the needed dataset. Using this dataset, we designed machine learning incast completion time prediction models using random forest regression. An analytical model for incast performance prediction is presented in Section 5.4. Evaluation results and analysis are provided in Section 5.5. We discuss related work in Section 6.6. Finally, Section 6.7 conclude the chapter.

5.1 Incast System Setup and Notations

Incast traffic is a many-to-one communication pattern in data centers where a large number of servers communicate simultaneously with a single client, as introduced in Chapter 4. It is present with DCN applications such as distributed storage, web-search with partition/aggregation design pattern, MapReduce, etc. Incast can cause severe congestion in switches and result in TCP throughput collapse, substantially degrading the application performance. The catastrophic TCP throughput collapse is explained by the fact that the bottleneck switch buffer is overfilled quickly as the number of competing senders increases. The overfilled buffer leads to packet losses and subsequent retransmissions after timeouts [Phanishayee 2008, Vasudevan 2009, Chen 2009]. The TCP retransmission timeout (RTO) is computed dynamically based on experienced RTTs, but it is subject to a configuration minimum RTO, RTO_{min} of around hundred of milliseconds (e.g., 200ms). This default value is orders of magnitude too large for data center environments where RTT is in the 10s or 100s of microseconds.

This challenging traffic pattern handling is critical for Datacenters. Several solutions were proposed to mitigate the TCP throughput collapse in the incast scenario. Most of them concern RTO_{min} tuning to adequate small values in the RTT scale [Chen 2009, Vasudevan 2009, Chen 2015]. Another approach consists of using Explicit Congestion Notification (ECN) marking at the bottleneck switch level to ensure that senders are quickly notified of the queue overshoot and then adjusting their sending rate accordingly [Alizadeh 2010b]. This approach prevents buffer overflow and subsequent timeouts. The work in [Xu 2019] proposes an intelligent selective packet discarding at the switch level. This intelligent discarding ensures that the sender responds to packet loss using fast retransmission/fast recovery instead of RTO, avoiding RTO's penalty.

Figure 5.1 shows a simplified topology of a typical incast scenario without loss of generality. N servers send each other the quantity SRU (Server Request Unit) simultaneously to the sink node. This scenario corresponds to the Fixed Fragment Workload (FFW) in contrast to the Fixed Block Workload (FBW), where the total

block size is fixed and partitioned amongst an increasing number of servers. We consider the setting parameters as in TABLE 6.1. These notations hold for the rest of this chapter.

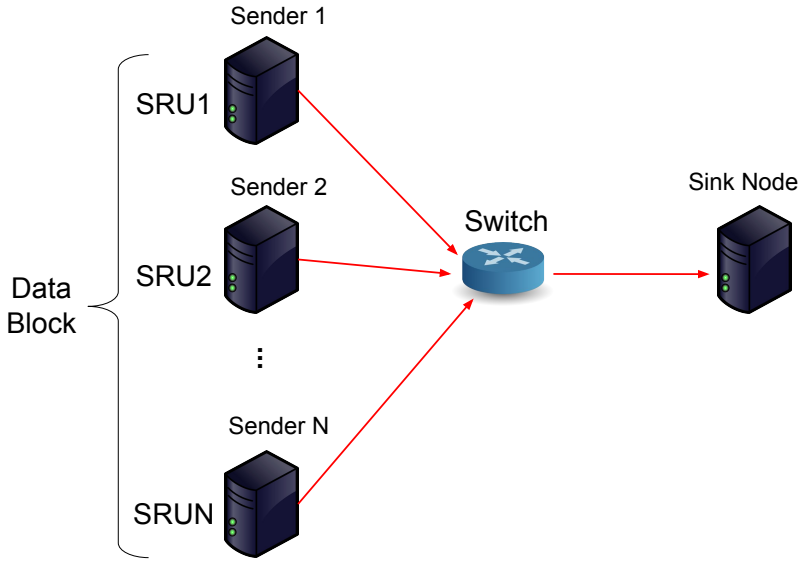

Figure 5.1: Simplified topology for a typical incast scenario

Table 5.1: Parameters and Notations

Parameters	**Description**
N	Number of competing senders
SRU	Server Request Unit size, per sender. SRU = 256 KB
B	Switch buffer size in packets. e.g. 64 pkts or 96 KB
C	Bottleneck link capacity. C = 1 Gbps
RTT_{noLoad}	RTT without queuing delay. $RTT_{noLoad} = 200\mu s$
RTO_{min}	Minimal TCP Retransmission timeout. e.g. 10 ms
S	TCP segment size, S=1446 bytes. Packet size = 1.5 KB
τ	Overall Incast Completion Time

5.2 SDN-enabled Machine Learning Incast Performance Prediction Framework

By following the typical Machine learning workflow for networking as specified in [Wang 2017] and leveraging SDN [Feamster 2014, Foster 2020], we come up with the SDN-enabled machine learning incast prediction framework in Figure 5.2. Indeed, SDN is already deployed and used in data center environments [Singh 2015]. The machine learning workflow for networking is very similar to the traditional machine

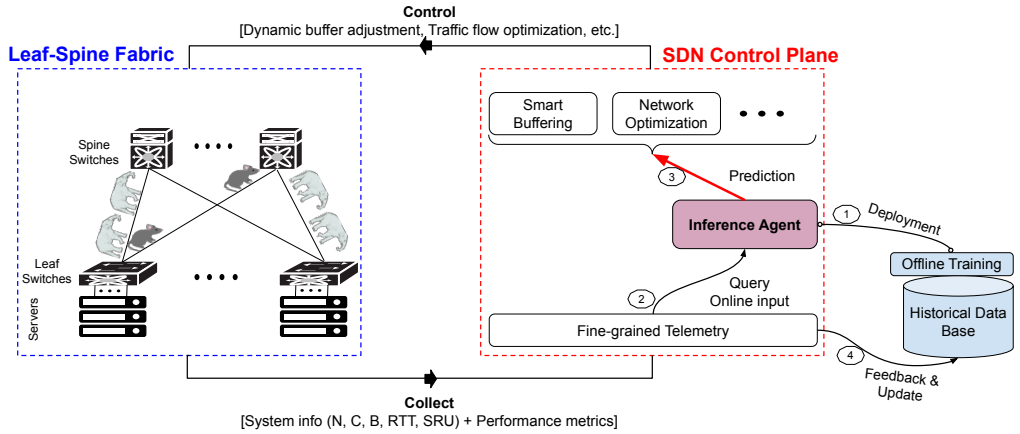

Figure 5.2: SDN-enabled Learning-based Incast Performance Inference Framework

learning one. It includes six stages as follows: Problem formulation, Data Collection, Data Analysis, Model Construction, Model Validation, and the last stage is Deployment and Inference.

The framework is based on two main cornerstones: SDN and the power of suitable machine learning algorithms of being able to learn some properties of a historical dataset and leverage the learned proprieties to provide good estimations on new observations.

From Figure 5.2 the workflow of the framework is as follow. Firstly the prediction model is constructed offline by doing training and parameter tuning on the historical data. The historical dataset may be composed of a large number of samples. Each sample represents a combination of features' values and the associated target value since we are in a supervised learning configuration. The features include the congestion algorithm used (tcpCC), the queuing discipline at the switch level (qdisc), the number of competing senders (N), the bottleneck bandwidth (C), the round-trip-time (RTT), the server request unit (SRU), the minimum retransmission timeout (RTO_{min}) and the target attribute is the incast completion time (τ). Prior knowledge or "domain-specific knowledge" and insights may be leverage at this stage of model construction.

The constructed model is then deployed (1) as the Inference Agent. Care should be taken for selecting the model concerning some operational aspects such as prediction latency, stability, and accuracy of the inference, etc. The model is deployed to be used for incast performance inference. Here real-time inference is desirable. Incast may generally consist of short flows which last few microseconds. If inference on real-time input could be done in real-time too, optimization or adjustments could be done before the incast payload transfer take really place. Otherwise proactive approaches could be used.

The online input (2), composed of (tcpCC, qdisc, C, SRU, N, RTT, SRU, RTO_{min}), is got when an incast traffic is initiated by the client leveraging SDN fine-grained telemetry and In-band network telemetry (INT) and P4. Taking this

input, an inference of the incast traffic's performance is done (3). The information will then be used by SDN-side smart buffering module or any traffic flow optimization algorithm to achieve efficient performance for the incast traffic and the co-existing ones. The optimization may concern for example network utilization maximization, global low mean delay, etc.

Finally, when the incast traffic complete, its really observed performance metric is also collected efficiently and the historical dataset could be updated (4). Having the database up-to-date is important, and will allow taking into account new dynamics from the data center. When the database significantly changed, the model needs to be re-constructed and re-deployed.

The historical data gathering and online update of the historical data with the newly collected data form a base for our framework. The historical data could also be enriched from outside (other owned data centers, or just from the cloud). For the data gathering concern, we will take advantage of the fact that a data center operator (e.g. Amazon, Microsoft, Google, Facebook) holds diverse data centers from which data could be gathered and mutualized. Indeed, this data collection needs to be done smartly in order to have very representative data comprising the features of interest. when an incast request is initiated the corresponding bottleneck switch knows the number of servers (N) involved in the incast traffic. The available bandwidth C could be estimated with traditional monitoring tools or lightweight bandwidth estimation through low overhead byte counter collection as presented in Chapter 3. And with INT/P4 we could collect any other useful end-to-end information from the data plane.

5.3 Learning-based Modeling

Recalling the workflow from [Wang 2017] we begin this section with the problem formulation. For the incast performance inference, the target metric (completion time) being a continuous variable, its prediction is a regression problem. It falls under the class of supervised learning algorithms. The other main classes being unsupervised learning and reinforcement learning algorithms.

5.3.1 Dataset and Analysis

We conducted intensive NS-3[1] simulations using the scenario topology in Figure 5.1 and varying the parameters from TABLE 6.1. For every simulation, we compute the corresponding completion time. We finally come up with a dataset composed of 46581 observations, six parameters, and one target variable, the completion time. The variables include two categorical variables: the congestion control algorithm used (NewReno or DCTCP) and the associated queuing discipline (FIFO or FQ_CoDel for NewReno and RED-ECN for DCTCP). The numerical variables are the bottleneck link bandwidth C, the round trip time RTT, the switch buffer

[1] https://www.nsnam.org/

size B, and the number of simultaneous senders N. The server request unit SRU and RTO_{min} were respectively fixed to 256000 bytes, and 10ms and are not part of the dataset.

For the dataset preparation for training, we consider two possibilities: a single model taking six features (two categorical and four numerical variables) and the case where we consider three different training sets for the different categories (NewReno_FIFO, NewReno_FQ, and DCTCP_RED). For the individual models' case, we use then only the numerical variables as training features. TABLE 5.2 summarizes information about the different datasets used.

We then consider these two cases in the data pre-processing step. We scale our data by standardizing numerical features. It consists of centering a feature's observations to the mean and scaled it to unit variance. For the singe model, we encode the two categorical features as a one-hot numeric array. Indeed five new numerical (binary) features are created to represent the categorical features' values (NewReno, DCTCP, FIFO, FQ_CoDel, RED-ECN).

Table 5.2: Datasets

Models	**n_samples**	**n_features**
Single Model	46581	6
NewReno_FIFO	15492	4
NewReno_FQ	15502	4
DCTCP_RED	15587	4

5.3.2 Model Training

After data analysis, we first investigate classical machine learning algorithms from less complex to the more complex without hyper-parameter tuning, in order to pick the most promising to work with. The models investigated are Linear Regression (lasso and ridge), Support Vector Regressor (SVR) with three kernels (linear, rbf, and polynomial), Decision tree, Random Forest(RF), and Multi-layer Perceptron (MLP). Random Forest only provides good results. Apart from decision trees, the other investigated machine learning algorithms were unable to capture the dataset dynamics, providing bad results. We then focus on Random Forest for the rest of this work, and as a proof-of-concept implementation. The machine learning algorithms are implemented using Scikit-learn 0.23.2 [Pedregosa 2011].

Random Forest falls under machine learning averaging methods, that combine the predictions of several base estimators here decision trees. The combined estimator is usually better since its variance is reduced. Decision trees are a non-parametric machine learning algorithm that predicts by learning simple decision rules inferred from the data features.

A random forest regressor has several hyper-parameters that need to be tuned for performance optimization. Some of the most important includes, the number of estimators (trees) used to construct the forest (n_estimators), the maximum

number of features provided to each tree (max_features), max_depth which depth we want every tree in the forest to grow, etc. For example, after a certain number, increasing the number of trees has almost no accuracy improvement but just increases model complexity with high training time.

Scikit-learn provides two main tools for hyper-parameter tuning GridSearchCV and RandomizedSearchCV. GridSearchCV exhaustively considers all parameter combinations from a parameter grid. On the other hand, RandomizedSearchCV can sample a given number of candidates from a parameter space with a specified distribution which is more convenient when we have a large search space. We use both on a set of parameter ranges but the Scikit-learn default hyper-parameter values perform quite well. The random forest regression algorithm with 100 estimators (trees) is used for both the single model case and the individual models one.

5.4 Analytical Modeling

Before presenting the evaluations results of our machine-learning performance prediction approach, we present here an analytical model for predicting incast completing time when TCP NewReno is used. Timeout is the main factor of goodput degrading [Zhang 2011]. We use recommended small RTO_{min} in the milliseconds, which solves quite acceptably the goodput collapse, making the timeout impact almost negligible. This analytical model is compared to the machine-learning-based in the next section.

5.4.1 Assumptions

Firstly we consider that the simultaneous incast senders are fully synchronized. The overall congestion window evolution follows an aggregate AIMD. This phenomenon is called TCP Synchronization, where multiple TCP connections increase and decreasing their congestion windows simultaneously. Then all the senders will be considered as a single aggregate source sending the total data to the client.

Secondly, we consider TCP congestion steady-state. Taking a macroscopic view of the traffic sent by the aggregate source, we can ignore the slow start phase. Indeed, the connection is in the slow-start phase for a relatively short period because the connection grows out of the phase exponentially fast. When we ignore the slow-start phase, the congestion window grows linearly, gets chopped in half when loss occurs, grows linearly, gets chopped in half when loss occurs and so on.

It's worth pointing out that one RTT is required to initiate the TCP connection. After one RTT the client sends a request for the incast data. The first bytes of the data are piggybacked onto the third segment in the three-way TCP handshake. After a total of two RTTs the client begins to receive data from the aggregate source.

5.4.2 Modeling Completion Time of Incast

Considering the assumptions mentioned above and being inspired by an existing model[2] we propose incast completion time analytical model as follows.

Let $X = \frac{N*SRU}{S}$, the number of segments present in the incast data. Using TCP and its AIMD congestion mechanism, we have the evolution of the congestion window as follows. The first window contains 1 segment, the second window contains 2 segments, the third window contains 4 segments, and so on. More generally, the k-th window contains 2^{k-1} segments. Let K be the number of windows that cover incast data to be transmitted. K can be expressed in terms of X as follows:

$$K = min\{k : 2^0 + 2^1 + 2^2 + ... + 2^{k-1} \geq X\}$$

$$K = min\{k : 2^k - 1 \geq X\}$$

$$K = min\{k : k \geq \log_2(X + 1)\}$$

$$K = min\{k : k \geq \log_2(\frac{N * SRU}{S} + 1)\}$$

After transmitting a window's worth of data, the server may stall (i.e., stop transmitting) while it waits for an acknowledgment. But not every time. Let us now calculate the amount of stall time after transmitting the k-th window. The time from when the server begins to transmit the k-th window until when the server receives an acknowledgment for the first segment in the window is $\frac{S}{C} + RTT$. The transmission time of the k-th window is $\frac{S}{C} * 2^{k-1}$.

The stall time is the difference of these two quantities:

$$max\{(\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}), 0\}$$

The server can potentially stall after the transmission of each of the first K-1 windows. (The server is done after the transmission of the K-th window.) We can now calculate the latency for transferring the overall incast data. The latency has three components: $2RTT$ for setting up the TCP connection and requesting incast data; $N * SRU/C$, the transmission time of the overall data; and the sum of all the stalled times. Thus, the incast completion time τ is:

$$\tau = 2 * RTT + \frac{N * SRU}{C} + \sum_{k=1}^{K-1} max\{(\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}), 0\}$$

We could obtain a more compact expression for the completion time with Equation 5.1 as follow:

[2]http://www2.ic.uff.br/~michael/kr1999/3-transport/3_07-congestion.html

$$\tau = 2 * RTT + \frac{N * SRU}{C} + \sum_{k=1}^{P} (\frac{S}{C} + RTT - 2^{k-1} * \frac{S}{C}) \tag{5.1}$$

With $P = min\{Q, K-1\}$
where $K = min\{k : k \geq \log_2(\frac{N*SRU}{S} + 1)\}$
and $Q = max\{k : k \leq \log_2(1 + \frac{C*RTT}{S}) + 1\}$

This model could be refined, approximating loss rate and including the corresponding retransmission times. However, these approximations are challenging. And with the machine learning approach, there is no need to look for such approximations. They are automatically learned from the data.

5.5 Validation and Analysis

Evaluation experiments were carried out on an Intel Core i7-7500U CPU 2.70 GHz x 4 with 16 GB of RAM running Ubuntu 16.04 LTS. We consider three evaluation metrics. The first is the prediction **score** (Eq. 5.2). It represents which proportion of the variance in the dependent variable is predictable from the independent variables. A more precise regression is one that has a relatively high R squared, close to 1. We will represent the score in percentage. Secondly we will use **NMAE** for Normalized Mean Absolute Error (Eq. 5.3). We want the NMAE to be as small as possible. And finally, we will consider the prediction time.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{\sum_{i=1}^{n}(y_i - \bar{y})} \text{ , with } \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{5.2}$$

$$NMAE(y, \hat{y}) = \frac{\frac{\sum_{i=1}^{n}|y_i - \bar{y}|}{n}}{\bar{y}} \text{ , with } \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{5.3}$$

5.5.1 Prediction Score and Normalized Mean Absolute Error

The first presented results concern prediction accuracy represented by the prediction score and the NMAE, all in percentage. Figure 5.3 shows prediction score and NMAE for different training size ratios for the single model and the individual ones. The general tendency is that the precision is quite stable with training ratios from 0.2 to 0.4 meaning a training set of 80% to 60% respectively. More tightly we can observe a slight decrease but not meaningful for the single model from 90.75% to 89.15%. The NMAE involves inversely with the general stability observed. The NMAE for the single model is around 20%.

The other observation is that the individual models perform better than the single model especially for NewReno_FIFO (97.78% to 97.03%) and NewReno_FQ (96.23% to 95.73%). The NMAE for NewReno_FIFO is around 7% and 8% for NewReno_FQ. However, performances are less good for DCTCP_RED where we observe score from 83.16% to 86.21% with the NMAE around 27%. Dynamics with

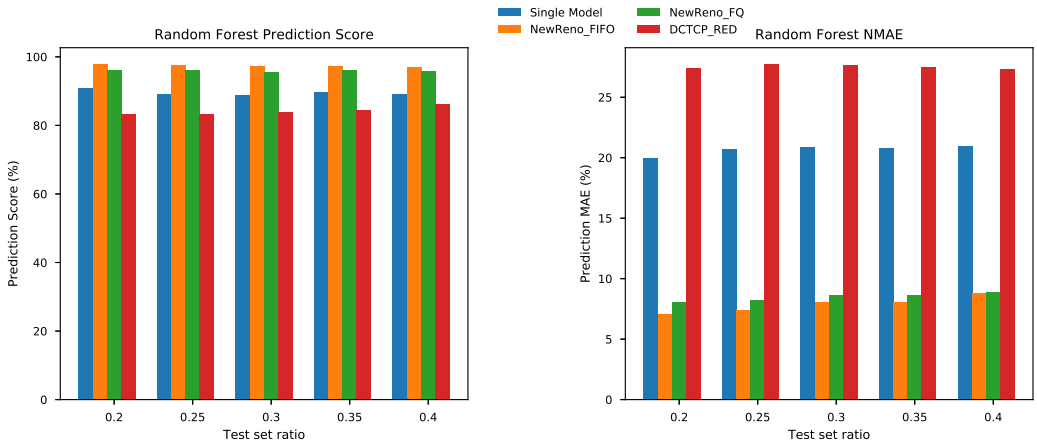

Figure 5.3: Score vs. Test size ratio

DCTCP dynamics is then more complex to capture, needing the investigation of other machine learning models or adding new features to improve its performance.

5.5.2 Machine learning vs. Analytical Model

Figure 5.4 presents some simulation data-points from the test set, their corresponding prediction with the individual random forest model, and the prediction with the analytical model presented in Equation 5.1. We present the results for NewReno with both FIFO and FQ_CoDel. The machine learning prediction follows well the data-points. The analytical model even in a simple form is able to capture the data-points apart from the points where the completion time is quite high. The normalized mean absolute errors for these shown data points are presented in TABLE 5.3 (where CC stands for the congestion algorithm used and QDISC, the associated queueing discipline).

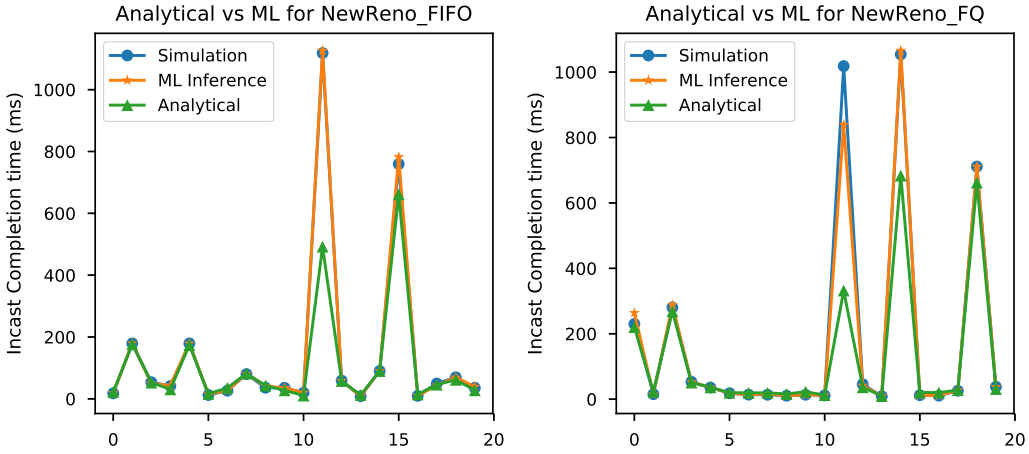

Figure 5.4: Learning vs. Analytical Modeling

Table 5.3: NMAE Analytical vs. ML

CC & QDISC	NewReno_FIFO	NewReno_FQ
NMAE ML	2.12 %	6.73%
NMAE Analytical	28.40%	33.22%

It's worth pointing out that the queuing discipline was not really taken into account during the model construction, at least explicitly. But when supposing overall synchronization of congestion windows that assume implicitly fair-queue share making the model suitable for fair queuing. Also, we note that fair queuing does not improve consequently the overall completion time. However, the bandwidth is fairly shared between senders, which is not the case with FIFO. With FIFO, some senders may finish transmitting their SRU very quickly and others too late presenting great unfairness between senders.

5.5.3 Prediction Time Distribution

Finally we present the atomic (one-by-one) prediction latency in Figure 5.5. The prediction time of the single model is slightly higher than those of the individual models since it is more complex and is constructed using all the individual training sets. However, this difference needs to be balanced with the fact that in the case of the individual models a prior process time is needed, to match input to the corresponding model.

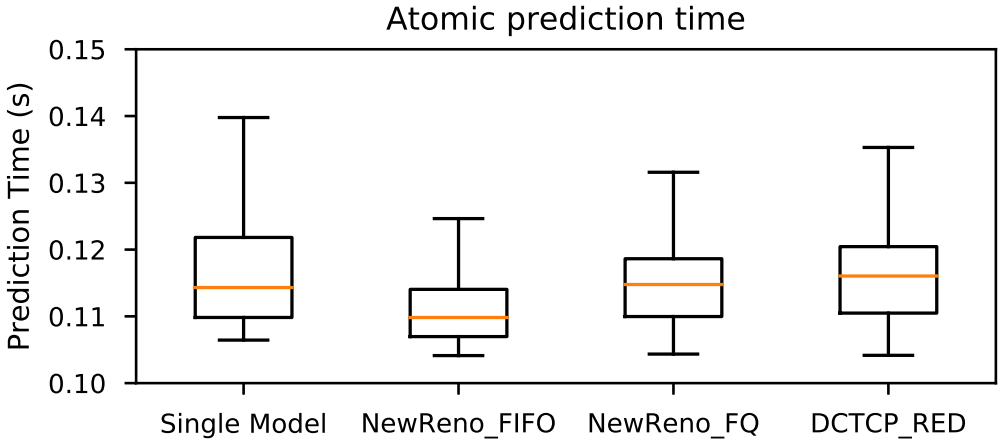

Figure 5.5: Atomic Runtime Prediction Latency Distribution

The prediction time is unfortunately high, around 0.10 seconds. This high latency can be explained by the use of Scikit-learn. Indeed, Scikit-learn is not necessarily suitable for production model deployment but more suitable for prototyping. Scikit-learn implements some methods in C for performance but additional overhead is present due to Python function calls, feature extraction, to name a few. Suitable input data format usage could improve performance and also bulk

prediction (many instances at the same time). Indeed when predicting bulk test sets, we have quite the same prediction latency as for atomic prediction and then the prediction throughput increases. The performance gain with bulk prediction can be explained with these factors: linear algebra libraries optimizations, branching predictability, CPU cache, etc. It's also pointing out that existing optimization solutions for machine learning pipelines are generally dedicated to the training step.

In production, more optimized implementations and frameworks coupled with specialized hardware (FPGA, GPU, TPU, Xilinx, etc.) are needed [Crankshaw 2019]. These hardware accelerators include FPGA (used for example by Microsoft and Xilinx ML Suites), Nvidia's GPU, AI ASICs (e.g. Google's TPU), etc. For our proposed inference system, we hope there will be enough computation resources in data centers management and control planes and the use of dedicated hardware will improve its performance. This way advantages of this machine learning inference could be effectively beneficial for overall flow QoS optimization in data centers.

Moreover, the use of the proactive approach where the control plane simulates what-if-scenarios by exploring some incast setups and parameter adjustments taking the prediction of the inference agent can help. Parameter adjustments needed to achieve global performance can then be anticipated. In this case, the prediction time penalty will be less severe.

5.6 Related Works

5.6.1 TCP Incast Modeling

The authors in [Chen 2009] analyze the dynamics of the incast problem by exploring its sensitivity to various system parameters. The understanding of the dynamics of incast is done with an analytical model based on empirical data. This quantitative model is completed with a qualitative refinement to capture most of the incast aspects, unfortunately, not all. This work, however, explains the root cause of incast, the RTO, and supports the TCP-level solution consisting mostly of using small RTO_{min} values in the data center RTT scales to alleviate throughput collapse.

The work [Zhang 2011] provides an analytical goodput model of incast where the goodput deterioration is explained by 2 types of timeouts. The block tail timeout observed when the number of simultaneous senders N is small, and the block head timeout when N is large. This work considers the sending of consecutive data blocks. The analytical model characterizes well the general tendency of TCP incast problem. This helps to understand the problem and helps understanding possible solutions such as RTO_{min} reduction. But when a new solution is provided to handle incast traffic, we may need to rebuild a new model to express attended performances.

Finally, authors in [Chen 2015] provide an in-depth understanding of how TCP incast problem happens with an interpretive model. This model explains qualitatively how systems parameters (block size, link capacity, buffer size) and mechanism variables (RTO_{min}) impact TCP incast.

5.6.2 Machine Learning for QoE/QoS inference in SDN

A comprehensive survey on machine learning algorithms' application to SDN can be found in [Xie 2018]. This application to SDN is guided by diverse objectives that include traffic classification, security, resource management, routing optimization, and finally Quality of service (QoS) / Quality of Experience (QoE) prediction. For this latter let us focus on two works [Pasquini 2017] and [Jain 2016].

An end-to-end application QoS prediction is proposed in [Pasquini 2017]. OpenFlow per port statistics are used to infer the service-level QoS metrics such as frame rate or response time for video-on-demand applications. Two machine learning algorithms (decision tree and random forest) are used.

The authors in [Jain 2016] propose a two-phase analysis approach for QoS inference, able to predict traffic congestion. Firstly it discovers which key performance indicators (KPIs) are correlated with the QoS metric using a decision tree. Then it mines each KPI's quantitative impact using linear regression.

The work in [Rusek 2020] proposes RouteNet that leverages the ability of Graph Neural Networks (GNN) for network modeling and optimization in SDN. Taking as input network topology information, routing schemes, and traffic matrix RouteNet, based on Generalized Linear Models, can provide accurate source-destination KPIs delay distribution (mean delay and jitter) packet drops prediction. These KPI predictions could be leveraged by QoS-aware optimizer to improve global performance.

5.7 Summary

In this chapter, we propose an SDN-enabled machine learning incast performance prediction framework for data center networks. This framework's goal is to provide incast completion time inference at run-time. This information could then be leveraged by any flow optimization algorithm or adaptive smart buffering mechanism to dynamically adjust system parameters to achieve efficient performance for both incast traffic and the co-existing traffic (generally elephant flows). We conduct intensive NS-3 simulations and construct a representative dataset. After that, a random forest regression model was implemented.

The evaluation results show that the proposed learning-based incast performance inference can provide good predictions using a single model or individual models depending on the congestion control algorithm and the queuing discipline used. We achieve up to 90% of prediction performance score for the one single model case, 97% for TCP New Reno with FIFO, 97% for NewReno with FQ_CoDel, and 86% for DCTCP with RED and ECN. We also compared our random forest model to the analytical model approach. The machine learning approach has the advantage of being easily generalizable for diverse congestion control and queuing discipline schemes, dynamic environments and of not being tightly coupled with any domain-specific assumptions and approximations. In Chapter 6, we will see how we can leverage this ML performance modeling to optimize data centers traffics through smart switch buffer management.

Chapter 6

ML-based Traffic Performance Optimization in SD-DCN

In this chapter, we leverage the ML-inference model from the previous chapter to effectively handle incast and elephant traffics in DCN. We define a performance optimization problem to find the best buffer-management-related parameters (switch buffer size and Active Queue Management (AQM) parameters) that achieve optimal performance for incast and elephant traffics in DCN using the ML model. On the first hand, we propose an exhaustive search procedure using the *generate and test* basic algorithm to optimize performance when considering only switch buffer sizing. And finally, for a more general and smart search algorithm, we leverage *Bayesian Optimization* with the ML-based inference framework to select the best or near-best buffer-management-related parameter settings to provide overall great performance to mixed incast and elephant traffic in DCNs.

We structure the chapter into seven sections. First, in Section 6.1, we present a mixed incast and elephant traffic scenario and the corresponding notations. We also formalize the smart switch buffering scheme as an optimization problem. In Section 6.2, we present the smart buffering framework and provide detailed information on its workflow. Section 6.3 recalls the ML modeling step, which is quite

similar to the procedure in the previous chapter. Then we present the optimization process in Section 6.4. We, first hand, consider the case where we have to choose only the optimal buffer size. For this case, an exhaustive search algorithm seems to be acceptable. But with additional AQM parameters, the search space explodes and leads us to BO. Section 6.5 presents the evaluation results. Related works are discussed in Section 6.6. And we finish the chapter with a conclusion in Section 6.7.

6.1 Scenario and Problem Formulation

6.1.1 Mixed Elephant and Incast Traffic Scenario and Notations

For this study, we consider a dumbbell topology of mixed elephant and incast traffic as shown in Figure 6.1. All senders and clients are connected to the switches with a 1Gbps link. The bottleneck link between the two switches S1 and S2, has the bandwidth C. Incast senders are connected in a star shape to the switch S1. In this figure, N incast servers send each other the quantity SRU (Server Request Unit) simultaneously to the incast sink node linked to S2. In this scenario, the elephant traffic corresponds to a bulk transfer from the elephant source to the elephant receiver (e.g., background continuous file transfer or server migration). We consider the setting parameters as in TABLE 6.1. These notations hold for the rest of the chapter.

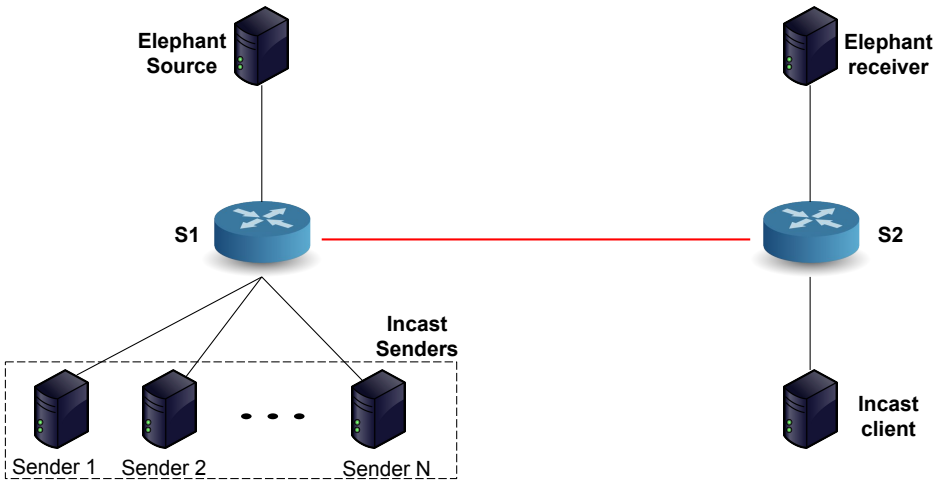

Figure 6.1: Basic topology of mixed elephant-incast scenario

Generally speaking, and especially in the scenario described above, congestion control algorithms (e.g., TCP Cubic through a classic FIFO qdisc on the switch level) fail to ensure good performance. Smart buffer management at the switch level composing of buffer sizing and AQM scheme is necessary to achieve better performance. Earlier investigations show that FQ-CoDel [Hoeiland-Joergensen 2018] is

Table 6.1: Parameters and Notations

Parameters	Description
N	Number of incast senders
SRU	Server Request Unit size, per sender; SRU = 256 KB
B	Switch buffer size in packets; e.g. 64 pkts or 96 KB
C	Bottleneck link capacity; e.g. C = 1 Gbps
$baseRTT$	RTT without queuing delay; e.g. $baseRTT = 200\mu s$
RTO_{min}	Minimal TCP Retransmission timeout; e.g. 10 ms
S	TCP segment size, S=1446 bytes. Packet size = 1.5 KB
CC	Congestion Control algorithm (e.g. Cubic)
qdisc	Queueing discipline (e.g. FIFO, Fq-CoDel)
τ	Incast Completion Time (s)
γ	Elephant Goodput (Mbps)

well suited for such scenario, and it provides very interesting results as also specified in this prior work [Gong 2018].

Indeed, FQ-CoDel for Flow Queue - Controlled Delay is a packet scheduler and an AQM algorithm developed to fight bufferbloat and reducing latency. It achieves this goal by reducing the impact of head-of-line blocking from bursty traffic and providing isolation for low-rate traffic. It was originally developed for home routers. Its default parameter values (e.g., target = 5ms and interval = 100ms) are then not suitable for data center environments with gigabit links and microsecond RTTs [Bufferbloat 2014]. To be effective to the mixed elephant and incast traffic, FQ-CoDel's parameters need to be tuned carefully and with well-defined switch buffer space.

The challenge for smart buffer management in this context is to select the best operating parameters (the switch buffer size B, the FQ-CoDel interval, and target) setting to achieve optimal performance for any inputs parameters combinations (represented by the congestion control CC used, the number of incast senders N, baseRTT, etc.)

6.1.2 Problem Formulation

Let $U(\tau, \gamma)$ denote the performance utility metric. It must satisfy the researched performance goal of providing high throughput (especially goodput in this study) γ for the elephant flow and low completion time τ for the incast traffic. Inspired from the network power metric [Floyd 2008, Winstein 2013], which follows a similar goal, we define $U(\tau, \gamma) = \log(\gamma) - \log(\tau)$.

Our objective is to maximize $U(\tau, \gamma)$ by finding adequate parameter settings. This goal is formalized with the optimization problem as follows.

$$\begin{aligned} \underset{P}{\text{Maximize}} \quad & U = f_X(P) \\ \text{subject to} \quad & P \in \Omega_P \end{aligned} \tag{6.1}$$

with P the set of operating parameters P_i (e.g., B, interval) with their respective values spaces (e.g., $B < B_{max}$). Ω_P is the resulting overall variables domains, not to say the optimization problem constraints. X represents the other parameters X_j (e.g., N, C, CC, etc.) that don't serve as decision variables of the optimization problem. These non-decision-variables will, however, be taken as inputs of the optimization model. Depending on implementation reasons or other concerns, we can consider the equivalent minimization problem as:

$$\underset{P}{\text{Maximize}} \quad U = f_X(P) \quad \Leftrightarrow \quad \underset{P}{\text{Minimize}} \quad -U = -f_X(P)$$

Formalized as in Eq. 6.1, the problem seems obvious to solve. Unfortunately, that is not the case. f expressing U based on the parameters P_i is an unknown function. Indeed, $\tau = g_X(P)$ and $\gamma = h_X(P)$ with g and h expressing the performance metrics τ and γ based on the parameters P_i and X_j are unknown functions. In other words, there is no analytical performance model for mixed incast and elephant traffic due to data center dynamics and complexity. Since $U = \log(\gamma) - \log(\tau) = f_X(P)$, f is also an unknown function. This makes our optimization problem challenging, and it falls in the category of the so-called black-box optimization.

To address this challenge, we will take advantage of the ML performance model studied in the previous chapter. Our proposed approach is integrated into the SDN-ML-DCN management framework. The resulted SDN-enabled ML-based smart buffer management framework is presented in the next section.

6.2 SDN-enabled ML-based Smart Buffer Management Framework

This smart adaptive buffer management framework (see Figure 6.2) can manage mixed incast-elephant traffic with overall better application performance in terms of efficiency (high elephant throughput and low incast completion time) and fairness. Indeed, by separating the network's control plane from its data plane, SDN introduces flexibility in network management, provides a global view of the network, and facilitates telemetry. Moreover, it eases the use of machine learning techniques in the management plane as presented in Chapter 5. Using machine learning, we could provide good predictions of the performance utility metric $U(\tau, \gamma)$ for different parameter combinations. The *Performance Optimizer* will leverage these predictions as f's estimates during the optimization process.

The workflow of the framework is quite similar to the one presented in Figure 5.2). Firstly, the performance prediction model is trained offline on the historical data. The samples of the historical dataset represent a combination of feature values and the associated target value since we are in a supervised learning configuration. The features include the congestion control algorithm used (CC), the queuing discipline at the switch parameters (e.g., FQ-Codel interval), the number of incast senders (N), the bottleneck bandwidth (C), the base round-trip-time (baseRTT),

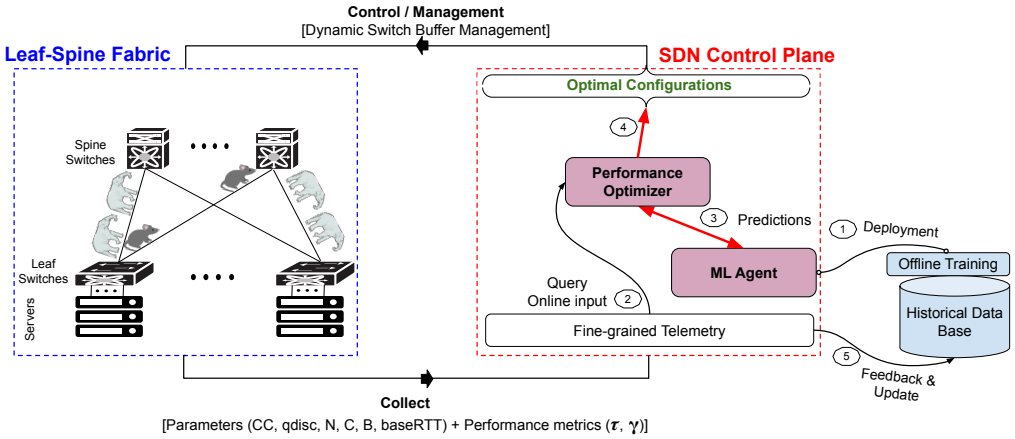

Figure 6.2: SDN-enabled ML-based Smart Buffer Management Framework

and the target attribute (the performance utility metric $U(\tau, \gamma) = \log(\gamma) - \log(\tau)$).

The trained model is then deployed (1) as the *ML Agent*. The online input (2), composed of (CC, qdisc, C, SRU, N, RTT, baseRTT), is got when an incast traffic is initiated by the incast client leveraging SDN fine-grained telemetry, In-band Network Telemetry (INT), and P4. Taking this input, The Performance Optimizer interacts with the ML agent by using the predictions as the utility observations (3). This interaction is repeated smartly till best operating parameters for the incast query are found (4). Ideally, the optimization process must converge in real-time. Otherwise, proactive management with future planning in what-if-scenario modeling will be used. With the optimal run-time parameters configuration, the system may observe overall efficient application performance.

Finally, at completion, real observed performance metrics (τ and γ are also collected, and the historical dataset could be updated (5). Having the database up-to-date is important, and will allow taking into account new dynamics from the data center. When the database significantly changed, the model needs to be re-constructed and re-deployed.

6.3 ML Performance Utility Prediction Model

The target value, the performance utility metric $U(\tau, \gamma)$, is a continuous variable. Its prediction is a regression problem and we are in a supervised learning case as in Chapter 5. For this ML-based prediction mechanism to be relevant for our optimization component, it needs to be simple and easily generalizable [Fu 2021]. Moreover, it is preferable if the model does not require a too high training set size to provide good predictions that capture well $U(\tau, \gamma)$'s dynamics.

The ML model construction for the simplified scenario of incast holds here. That simplified setup was well choosen with no lost of generality. We then directly leverage the results and lessons learned from that study of the previous chapter.

We use Random Forest as ML algorithm with Scikit-learn default hyperparameters (e.g., the number of trees n_estimators = 100).

6.4 Optimization Process

6.4.1 Preamble

Here, the goal is to find the optimal buffer-based parameter setting to provide overall better for both elephant and incast flows. Thanks to ML, these best parameters can adapt to the system changing represented by the parameters CC, C, baseRTT, etc. The manageable parameters would generally be related to the switches, such as its buffer space or its queuing discipline. Adaptive buffer management [Chuprikov 2020] can be achieved easily with data-plane programmability powered by P4.

By solving the problem formulated in Section 6.1 to provide optimal performance, the challenge is twofold:

Ideal Buffer Sizing. Indeed, buffer is in the heart of the vast majority of performance issues [hus 2019]. For internet backbone routers, the rule-of-thumb states a buffer size equals the output bandwidth C times the round-trip-time RTT $B = C * RTT$ (the BDP, bandwidth-delay product), to keep high utilization at the bottleneck link [Villamizar 1994]. With the increase of high-speed links, the BDP rule was improved by [Appenzeller 2004] that proposes a smaller buffer requirement $B = (C * RTT)/\sqrt{n}$, where n is the number of flows with almost the same performance. Unfortunately, when it comes to data center switches' buffer requirements, as far as we know, there is no rule-of-thumb. A widely standard recommendation is to use in data centers relatively small buffers to achieve high bandwidth, and low latency [Alizadeh 2010b, Raiciu 2019, Shpiner 2016]. But due to the critical aspect of buffer size on performance, buffer sizing needs more attention.

Best AQM (FQ-CoDel) Parameters. As mentioned earlier, FQ-CoDel, originally developed for home routers, provides interesting results for our mixed elephant incast traffic. However, its default parameters values (e.g. target = 5ms and interval = 100ms) are not suitable for data center environments with gigabit links and microsecond RTTs [Bufferbloat 2014]. To be effective in this environment, FQ-CoDel's parameters (interval and target) need to be tuned carefully and ideally in an automatic manner. The FQ-CoDel target parameter was fixed to 10% of the interval as recommended in [Hoeiland-Joergensen 2018]. We will then search for the best interval T value according to the situations.

6.4.2 Exhaustive-Search-based Optimization Algorithm

With a focus on the switch buffer size B ($P = \{B\}$), the optimization problem consists of finding the best B* that maximizes the performance utility metric U for a given combination of the other parameters X. The optimization procedure is done in three steps: (i) Generate a set of candidates B, (ii) Evaluate the resulting

Algorithm 3: Exhaustive ML-based Performance Optimization

Input: ML_model, maximum buffer space B_{max}
Output: Recommended optimal buffer space $B*$

```
1 Generate a search space Ω_B
2 for B ∈ Ω_B do
3 |   û ← ML_model(B)
4 |   U_tab[B] ← û
5 end
6 B* ← argmax_B U_tab
```

incast completion time of each of them, and (iii) selects the one that satisfies the optimization objective.

The algorithm procedure is presented in Algorithm 4. It begins by generating the set of candidates B to test Ω_B (line 1). For an incast traffic, the corresponding parameters are collected and the expected the utility metric U is inferred with the Random forest model (line 3) for various candidates B. After this evaluation, the best-fitted candidate B* (line 6) is used by the SDN controller to dynamically adjust the switch buffer space to handle efficiently the traffic.

6.4.3 Bayesian-Optimization-based Algorithm

Considering only the switch buffer space B as the decision variable of the optimization problem, the exhaustive search approach is tolerable. But when considering additional parameters (e.g., the FQ-CoDel interval T), the search space increases, and the exhaustive search that requires evaluating all possible candidates is no more practicable. We then propose to use Bayesian Optimization (BO) [Shahriari 2015] to solve the general smart buffering optimization problem in Eq. 6.1. BO is a framework to solve optimization problems like this one where the objective function f is unknown beforehand but can be observed through experiments [Alipourfard 2017]. Indeed, there are no gradients, and typically f evaluation is expensive, and the observations may be noisy. The optimization process involves designing a sequential strategy that maps collected data to the next query point to find optimal parameters. For our case, for a new query point to evaluate, the observation means to run effectively through the network the corresponding scenario. This is not possible, and we propose to rely on historically collected data. We will construct an ML prediction model from this data that can provide evaluation for any query point as prediction, even those that were not really observed yet.

The optimization problem consists of finding the best buffer-based parameters P (B* and T*) that maximize U for a given combination of the other parameters X_j. Since the framework could provide ML performance predictions, an obvious optimization procedure would consist simply of scanning throw all the candidates $P_i = (B_i, T_i)$ from Ω_P and selects the one that maximizes the objective. This procedure has a high overhead (the prediction time), but it will also propagate the

Algorithm 4: BO&ML-based Performance Optimization

Input: ML_model, Ω_P, $\mathcal{M}$, $\mathcal{A}$

Output: Recommended optimal parameter setting P*

1 $\mathcal{D}_1 \leftarrow$ Init_Samples(ML_model, Ω_P)

2 **for** $i \leftarrow$ *1, 2, ...* **do**

3 Prob(Y | P, $\mathcal{D}_i$) $\leftarrow$ Fit_Model($\mathcal{M}$, $\mathcal{D}_i$)

4 $P_{i+1} \leftarrow \underset{P \in \Omega_P}{\text{argmax}}\ \mathcal{A}(\text{P, Prob(Y} \mid \text{P, } \mathcal{D}_i))$

5 $Y_{i+1} \leftarrow ML\_model(P_{i+1})$

6 $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \cup (P_{i+1}, Y_{i+1})$

7 **end**

prediction errors through all the search space. To reduce the search time and overhead, we propose the smart search procedure presented in Algorithm 4 leveraging BO [Shahriari 2015, Dewancker 2015].

The general idea is to build a probabilistic surrogate model (e.g., Gaussian process) $\mathcal{M}$ that puts a prior belief over the possible objective functions, and sequentially refine this model since data (P_i, Y_i) are observed via Bayesian posterior updating. Let $\mathcal{D}$ denote the available data. The updated model is queried to select next candidate P_{i+1} in the search space through acquisition function $\mathcal{A}$ (e.g. UCB - Upper Confidence Bound) maximization. Indeed, the acquisition function trade off exploration and exploitation. It is high where the surrogate model predicts a high U (exploitation) and where the model prediction uncertainty is high (exploration).

Algorithm 4 falls under the class of sequential model-based optimization. Using a starting set of samples $\mathcal{D}_1$ from the parameter space Ω_P (*line 1*), the probabilistic regression model $\mathcal{M}$ is initialized (*line 3*). Then new candidates P_{i+1} from the search space are sequentially selected by optimizing the pre-defined acquisition function $\mathcal{A}$ (*line 4*). $\mathcal{A}$ uses the current probabilistic model as a cheap surrogate for the black-box and expensive objective $U = f_X(P)$, which is represented here by the random forest model ML_model. Each new P_{i+1} evaluation produces an observation Y_{i+1} (*line 5*) which is appended to the historical set $\mathcal{D}_i$ as $\mathcal{D}_{i+1}$ (*line 6*). The new set $\mathcal{D}_{i+1}$ will then be used to update the regression model $\mathcal{M}$ and new candidate suggestion will be done. All this process is repeated in respect to a so-called query budget (e.g. number of iterations) or till a certain stop condition is reached (e.g. the model confidence interval falls below a threshold).

At the end (e.g., after K iterations), a final recommendation of the best parameters P* (here B* and T*) are used by the SDN controller to dynamically adjust the switch buffer space and configure FQ-CoDel to handle the mixed incast elephant traffic efficiently.

For the algorithm implementation, we use Scikit-Optimize (or skopt) built upon Scikit-Learn.

6.5 Experimentation Study

We consider three evaluation metrics concerning the ML prediction accuracy. The first is the prediction **score**. It represents which proportion of the variance in the dependent variable is predictable from the independent variables. The most precise regression model would be the one that has a relatively high R squared, close to 100, when expressed in percentage. We will represent the score in percentage. Secondly, we use **NMAE**. We want the NMAE to be as small as possible. And finally, we will consider the **relative prediction error**: $\frac{|y_i - \hat{y}_i|}{y_i}$.

6.5.1 Exhaustive Search for BS only Experiments

6.5.1.1 Setup

We conducted intensive NS-3 simulations using the scenario topology in Figure 6.1 and varying the parameters from TABLE 6.1. For every simulation, we compute the corresponding completion time. We finally come up with a dataset composed of 83200 observations, six parameters, and one target variable, the incast completion time. The variables include two categorical variables: the congestion control algorithm CC used (NewReno or Cubic) and the associated queuing discipline qdisc (FIFO or FQ_CoDel). The numerical variables are the bottleneck link bandwidth C, the base round trip time $baseRTT$, the switch buffer size B, and the number of simultaneous incast senders N. When using FQ_CoDel the target parameter and and the interval T were fixed to the default values. The server request unit SRU and RTO_{min} were respectively fixed to 256000 bytes, and 10ms and are not part of the dataset features.

Another working hypothesis considered here is to only optimize the incast traffic's performance, considering the elephant one as not critical. meaning minimizing incast completion time τ. Then, the objective is now: $\underset{P}{\text{Maximize}} \;\; U = f_X(P) = -\log(\tau) \Leftrightarrow \underset{P}{\text{Minimize}} \;\; -U = \log(\tau) \Leftrightarrow \underset{P}{\text{Minimize}} \;\; \tau$.

For the dataset preparation for the model training, we consider two possibilities: a single model taking six features (two categorical and four numerical variables) and the case where we elaborate individual models for the 4 categories (NewReno_FIFO, NewReno_FQ, Cubic_FIFO, Cubic_FQ). We will consider individual models' case. Each category is represented by a dataset of 20800 observations with only the numerical variables. In the data pre-processing step, the dataset is scaled by standardizing these numerical features. It consists of centering a feature's observations to the mean and scaled it to unit variance.

From a minimalism perspective, for our ML model, we will not use all the samples from our dataset. A subset with n_samples (e.g. 5000, 10000, 15000, etc.) will be used after shuffling the entire dataset. Form this minimalist set, 70% will be used to train the model while the 30% remaining will be used to validate the trained model.

6.5.1.2 ML Prediction Accuracy

For a subset with n_samples of 10000, where 7000 samples are used to train the random forest model we obtain the scores and NMAE on the 3000 remaining set as presented in TABLE 6.2. Using all the dataset (20800 samples) generated through NS-3 simulations gives scores around 99% with NMAE of around 1%. These results prove the ability of the ML-performance approach to provide accurate predictions.

Table 6.2: ML predictions Accuracy

Categories	**Score (%)**	**NMAE (%)**
NewReno_FIFO	96.54	9.81
NewReno_FQ	96.95	9.07
Cubic_FIFO	96.90	9.26
Cubic_FQ	96.23	10.98

The prediction score and NMAE provide a good picture of the prediction accuracy. But they don't give detailed information on the model behavior. This information is provided in Figure 6.3 which presents the CDF (Cumulative Distribution Function) of the relative errors over all the evaluation samples. This distribution of residuals shows that the prediction error is very low for the majority of the test data-points for the 4 categories.

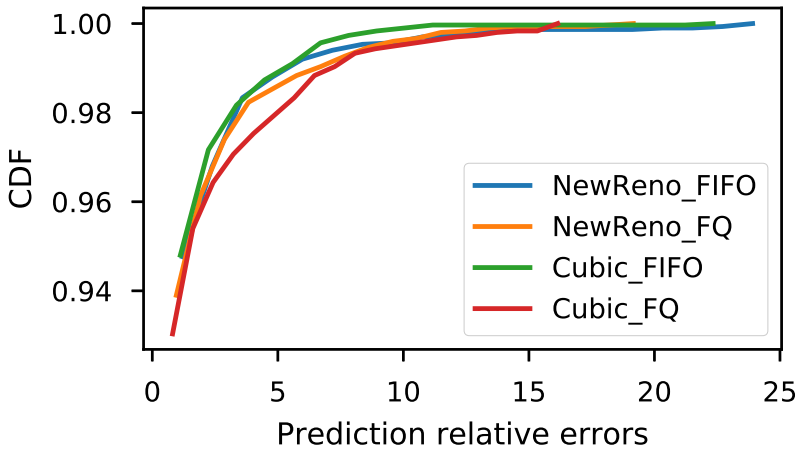

Figure 6.3: CDF of predictions relative errors

In Figure 6.4, we show For Cubic_FQ the real observed test points and their corresponding predictions with the random forest model. This regression plot confirms the significant relative errors observed from the CDF plot. Of course, when the subset contains all the initial dataset (K = 20800 samples), the prediction accuracy is better with a score almost equals to1 as shown in Figure 6.5.

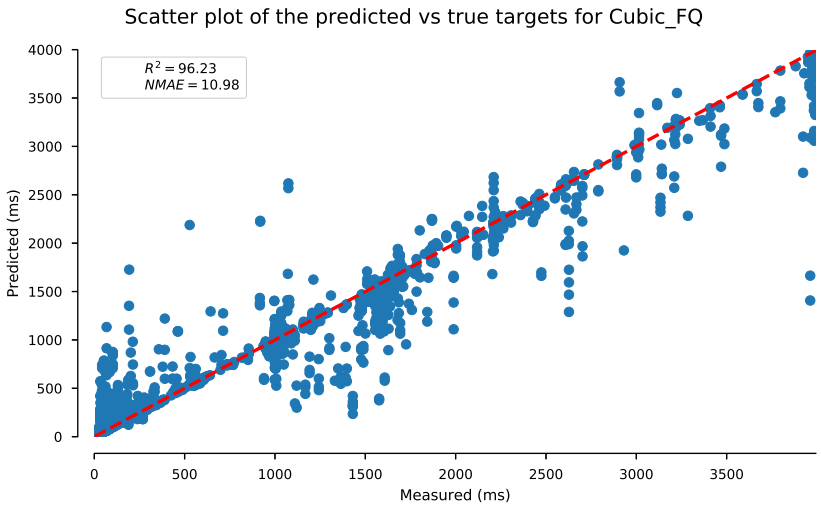

Figure 6.4: Predictions vs. Real observations (K = 10000 samples)

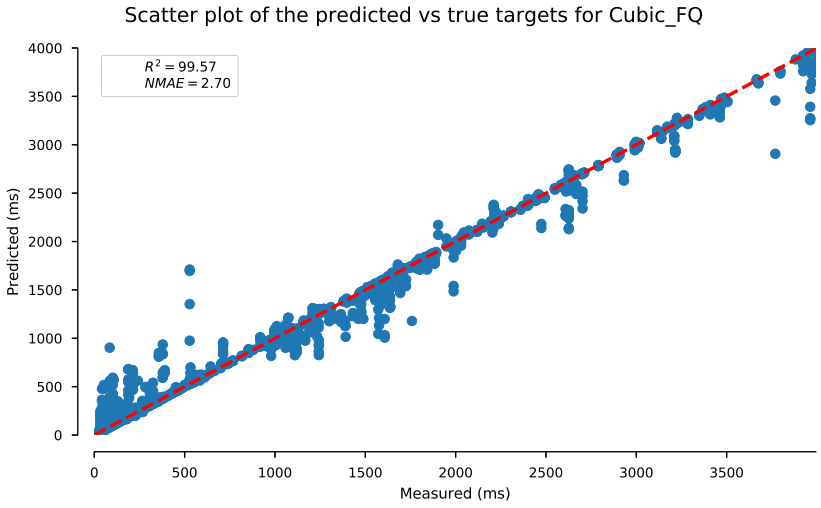

Figure 6.5: Predictions vs. Real observations (K = 20800 samples)

6.5.1.3 Optimization Evaluation Results

To evaluate the effectiveness of our optimization algorithm Algorithm 4, we compute B* for diverse values of N, with all the other parameters fixed. The algorithm output is presented in TABLE 6.4. We will compare the performance observed using this optimal B versus when using the maximum available buffer space B_{max}.

Table 6.3: Optimal Buffer space B* from Incast Performance Optimization Algorithm

N	1	4	8	16	20	32	48	64	100
B*	4	16	64	64	40	32	64	25	64

Figure 6.6 shows the incast completion time τ when using Algorithm 4 against over-provisioning using B_{max}. We can observe that τ is almost the same with both B* and B_{max}. It follows from these results that, with our framework, we could achieve great incast performance while preventing buffer wastage that may occur when over-provisioning. Moreover, sometimes, using the maximum available buffer space could degrade the performance (N = 32 on Figure 6.6).

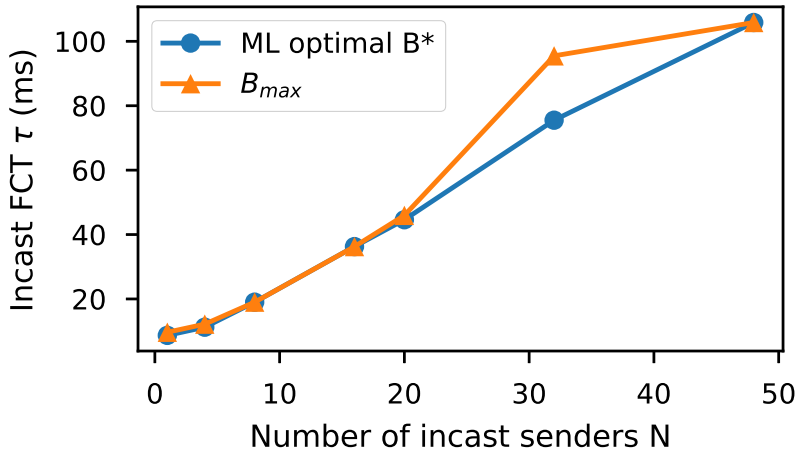

Figure 6.6: Incast FCT Optimization for diverse number of incast senders

6.5.2 BO for BS and AQM parameters tuning Experiments

6.5.2.1 Setup

As in the previous experiment, a working dataset is generated from intensive NS-3 simulations. These simulations use the mixed incast-elephant traffic scenario from Figure 6.1. For this study, we fixed the CC to Cubic, the qdisc to FQ-CoDel, C to 1Gbps, baseRTT to 100μs, SRU to 256000 bytes, and RTO_{min} 10ms. For every simulation, we compute the corresponding performance metric U. We generate a

dataset composed of 1690 observations, three features, and one target variable, the performance utility metric. The features are all numerical variables and include the number of simultaneous incast senders N, the switch buffer size B, and the FQ-CoDel interval parameter T. The server fixed parameters are not part of the dataset features used during the model training.

In Figure 6.7, we show the values of the utility metric in the search space. We can remark that it varies a lot with N. And with a given value of N, we can have multiple near-optimal zones. This great variability confirms the complexity[1] of the DCN traffic optimization problem, which calls for automatic ML-based modeling instead of an analytical one. More, this ML model may be coupled with intelligent Optimization algorithms. The exhaustive search is not an efficient approach for this case.

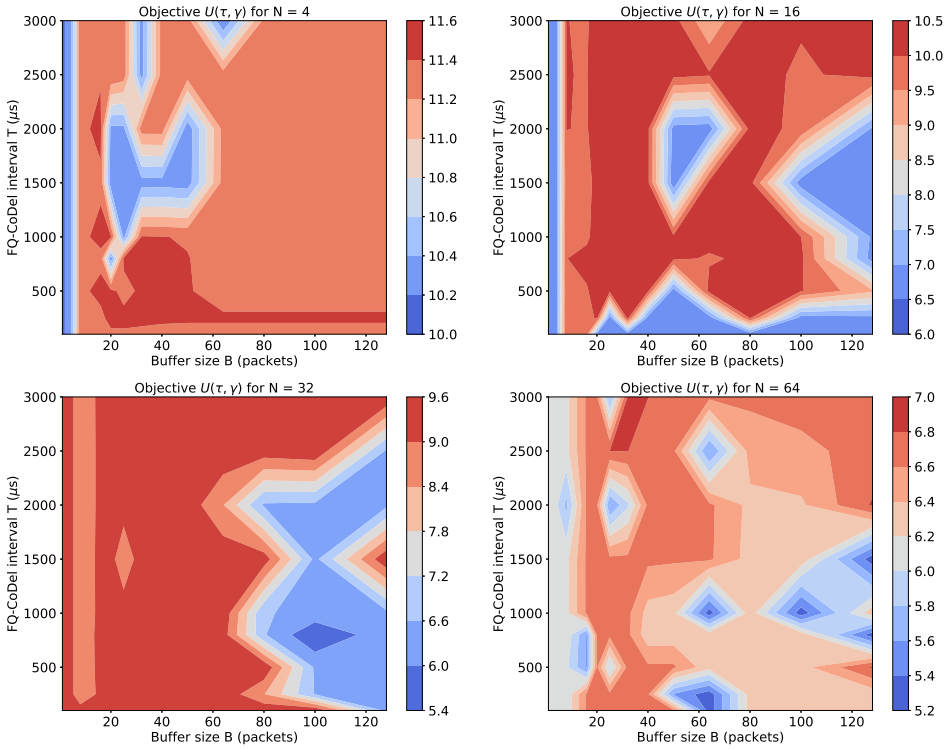

Figure 6.7: $U(\tau, \gamma)$ for different values of N

6.5.2.2 ML Prediction Accuracy

.

The random forest model shows a prediction score of 93.73% and an NMAE of 3.15%. The prediction statistics provide a good picture of the prediction accuracy, but they don't give detailed information on the model behavior. Figure 6.8 plotting scatters of real observed test points, and their corresponding predictions show

1

this individual behavior. These results prove the ability of the ML-performance approach to provide accurate predictions of the utility performance metric.

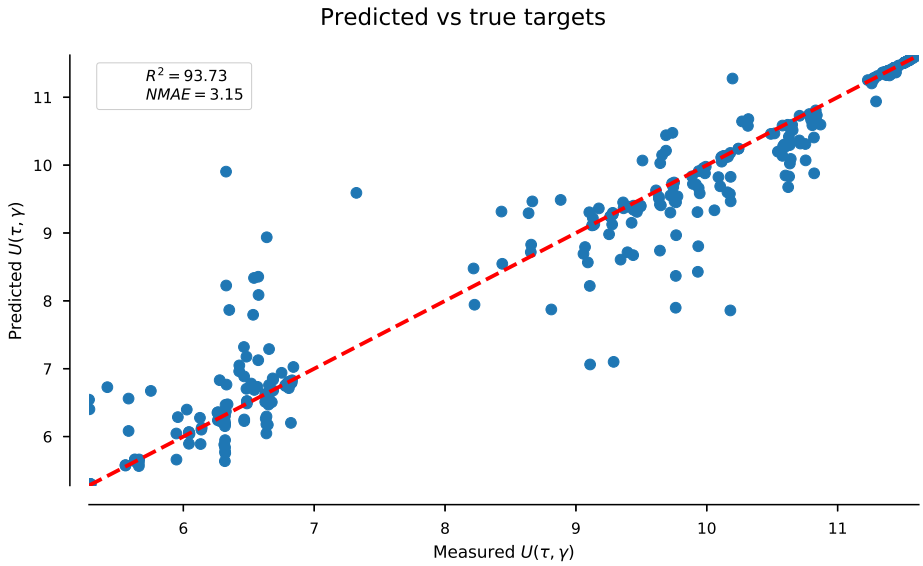

Figure 6.8: Predictions vs. Real observations

We can observe two regions. The first one has a high utility metric $U \in [9, 12]$ corresponding incast with relatively small incast senders (< 32 or 48) and the second one with metric $U \in [5, 7]$ for more increasing incast servers numbers. This observation highlighted the fact that with increasing incast senders number degrade performance for both incast and elephant. The objective of the smart buffering scheme is to provide performance close to the better possible one.

6.5.2.3 Optimization Evaluation Results

To evaluate the effectiveness of our BO algorithm Algorithm 4, we compute (B*, T*) for diverse values of N, with all the other parameters fixed. The algorithm output is presented in TABLE 6.4. We will compare the performance observed using these optimal parameters (B*, T*) U_{real} versus when using the maximum available buffer space (B_{max} and T_default) U_{dflt}.

Table 6.4: Optimal Parameters (B*, T*) from the BO&ML-based Algorithm

N	1	4	8	16	20	32	48	64	100
B*	1	16	64	38	35	29	61	35	45
T*	100	1250	2804	3000	2750	766	100	2750	571
U_{pred}	11.62	11.11	10.64	10.15	9.96	9.41	8.94	6.79	6.59
U_{real}	11.62	11.38	10.81	10.13	9.96	9.22	9.05	6.65	6.61
U_{dflt}	11.51	11.27	10.23	10.18	9.94	9.2	9.09	6.82	6.64

We can see that choosing the maximum available buffer size can provide almost the same performance as our approach from these results. However, this is a lazy and inefficient approach. Indeed contour plots of the performance utility metric show many near-optimal regions, and even a relatively small switch buffer with the best AQM parameters setting can achieve excellent performance. The BO&ML approach can then prevent buffer wastage that may occur when over-provisioning. And also, it is possible to select the best parameters when prioritizing between incast and elephant traffic. The utility metric would become $U(\tau, \gamma) = \log(\gamma) - \alpha * \log(\tau)$, where α guides this prioritization.

6.5.2.4 Optimization Algorithm convergence

Figure 6.9 shows the convergence of our BO-based performance optimization algorithm for different probabilistic models, and also we compare with a dummy random search algorithm. The Gaussian Process (gp_results) outperforms the other models. It converges in few iterations, less than 10. It's worth pointing out that the random optimizer performs very well too, this can happen, but there is no guarantee of convergence in general.

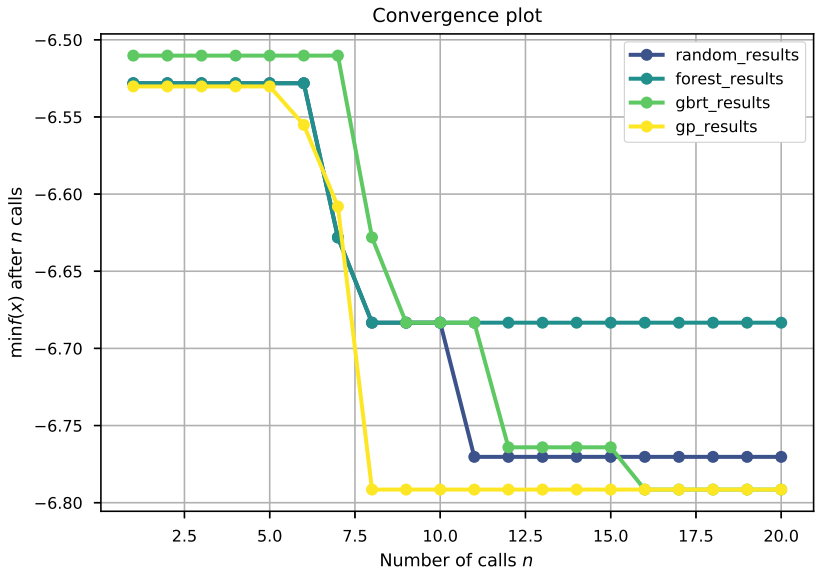

Figure 6.9: Convergence plot of Algorithm 4 For N = 64

We present again for N = 64, the candidates explored and exploited with the partial dependence of the buffer size B and the FQ-CoDel interval T in Figure 6.10. As additional information, Figure 6.11 precises the number of samples for each parameter of the search space as a histogram.

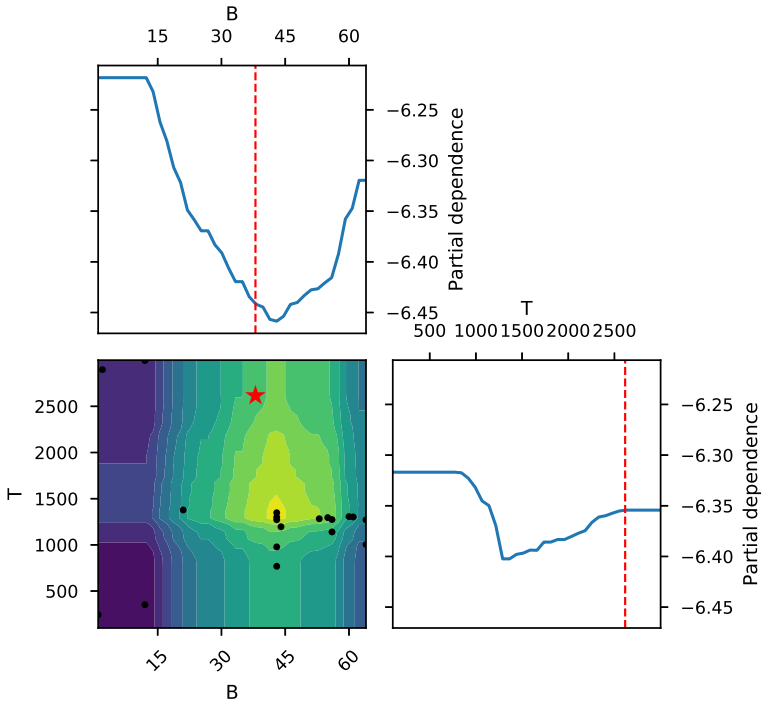

Figure 6.10: Objective U evaluation during Algorithm 4 execution

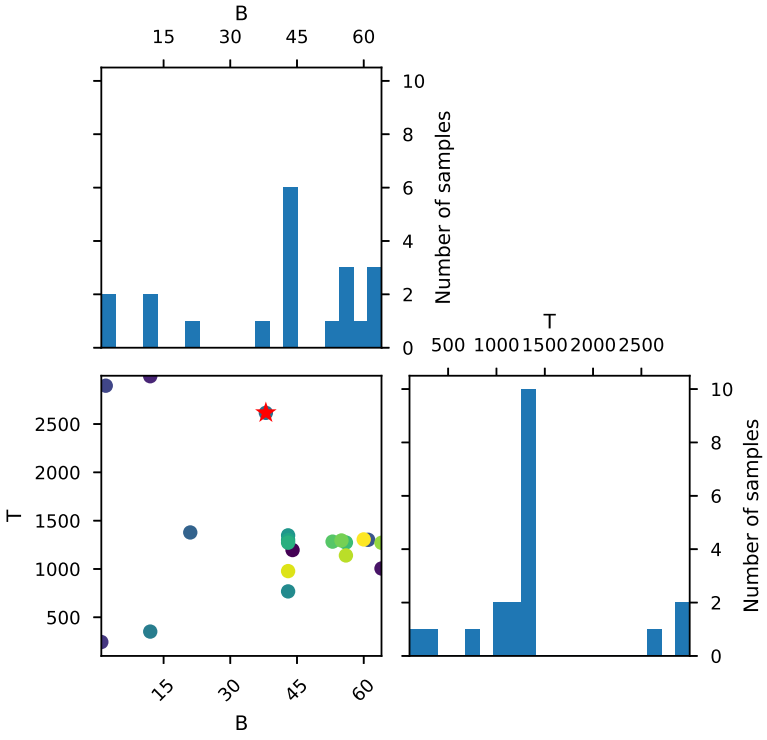

Figure 6.11: Number of Samples of the parameters with the Algorithm 4 execution

6.6 Related Works

Prior works have explored the use of ML to provide models for network and performance optimization. The work in [Rusek 2020] proposes RouteNet that leverages the ability of Graph Neural Networks (GNN) for network modeling and optimization in SDN. Taking as input network topology information, routing schemes, and traffic matrix RouteNet, based on Generalized Linear Models, can provide accurate end-to-end performance metric predictions such as delay distribution (mean delay and jitter) and packet drop prediction. These predictions could then be leveraged by QoS-aware global performance optimization.

The work in [Chen 2018] explores if Deep Reinforcement Learning (DRL) can be used for automatic traffic optimization in datacenters. Their preliminary study shows that The DRL approach's high latency is an obstacle to traffic optimization at the scale of current datacenters. Leveraging long-tail distribution of datacenter workload, they propose AuTO mimicking the Peripheral and Central Nervous Systems in animals to solve this scalability problem. Their work focuses on flow scheduling and load balancing. They adopt Multi-Level Feedback Queueing by optimizing its thresholds.

The study in [Fu 2021] evaluates whether ML offers a simple and general approach to performance prediction. It evaluates 6 ML performance prediction models across 13 real-world applications. The authors of [Fu 2021] show that many applications exhibit a surprisingly high degree of irreducible prediction error. And they propose a more nuanced methodology for applying ML for performance prediction.

6.7 Summary

This chapter explores DCN traffic performance optimization through smart buffer management. Firstly an exhaustive search algorithm is proposed to solve the resulted black-box optimization problem. It allows optimization of unknown or difficult to obtain analytical performance models by exploiting machine learning predictions. It is acceptable for a small search space (e.g., considering only buffer sizing). But when additional buffer-management-related parameters are also considered, the exhaustive approach is not an efficient solution. We then propose the SDN-enabled DCN management framework to leverage Machine Learning (ML) and Bayesian Optimization (BO), aiming to achieve optimal performance in data centers by finding the best switch buffer size and AQM parameters. It allows optimization on unknown or difficult to obtain analytical performance models of modern data centers applications. Evaluations based on intensive NS-3 simulations show that we can find the best or near-best buffer-management-related parameters (buffer size and AQM parameters) to handle mixed incast and elephant traffic with great performance. We expect this framework to be a building block for autonomous data center management.

Chapter 7

Conclusion

With new trends in networking (IoT, 5G, cloud-based data centers, etc.), modern and future networks constitute a complex and highly dynamic environment. This complexity makes network management difficult, and even networks seem unmanageable with traditional management schemes involving human intervention and dependence. Inspired by self-driving cars, self-driving networks constitute a new promising networking research topic leveraging SDN, machine learning, and automation[1] to remove humans from the management loop, at least partially, to address this magnifying complexity.

Network management teams aim to continuously deliver application and service performance and protection. Then, they need continuous network monitoring and optimization to support increasingly dynamic and digitally-driven business models. This goal is not achieved with SDN alone, nor with ML alone applied to network problems. However, it is achievable within the SelfDN vision that combines AI/ML on top of SDN-based networks. SDN Provides excellent operational flexibility and programmability in how to manage networks but introduces substantial challenges. AI/ML advances in computing power (e.g., GPU, TPU) and storage are expected to help cope with these challenges. AI/ML will drive network management by rapidly correlating information from multiple data sets to extract real-time insights to manage complexity (increased number of connections, programmability, etc.) and optimize resources. ML will complete the work opened by SDN. Figure 7.1 summarizes the journey from SDN to SelfDN and the pushing elements.

7.1 Summary of Contributions

Self-driving networking with the potential to simplify network management is a grand challenge for networking and the “Holy Grail” to reach for network management research. Even if some building blocks exist in the literature, the selfDN trip

[1] "Networks will operate as a system with increasing levels of autonomy, taking into account their own state, the dynamic state of all the users and applications, and the vast array of possible options", Ravi Chandrasekaran SVP, Enterprise Networking Business

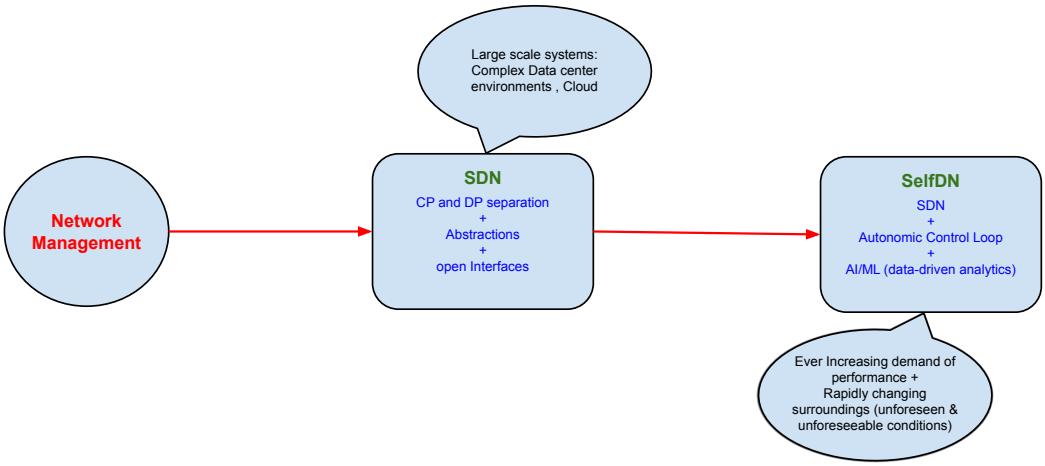

Figure 7.1: Network management: From SDN to SelfDN

has a long way to go since its deployment is almost nonexistent today. It, indeed, presents several challenges that need to be addressed to enforce potential adoption. In this thesis, we have studied two main challenges of the SelfDN vision: (i) The need for improved SDN monitoring; (ii) The need for more automation through the use of AI/ML for complex management tasks. The results of our contributions are summarized as follow:

Improved SDN Monitoring (Chapter 3). For our first contribution, we propose a generic monitoring framework for SDN to provide efficient data collection to construct the global and up-to-date view of the underlying data plane at the SDN controller level. This framework distinguishes event-based data collection, ad-hoc data collection with pull request/reply messages, and finally, periodic statistics (counters) collections. For this latter category, we are in the presence of a dilemma that concerns the optimal reporting interval time to use to achieve a good accuracy-overhead trade-off. Unlike the literature works that propose adaptive polling rate mechanisms, we propose a more efficient adaptive solution named COCO (**CO**nfidence based adaptive **CO**llection). COCO is a pushed and prediction based adaptive periodic data collection scheme. To provide accurate flow counters evolution with low overhead, COCO seizes these two following optimization opportunities:

- On the first hand, the pull model with request-reply messages of standard OpenFlow collection is inefficient. With a periodic collection or even with our adaptive collection, the collection points in time are known by the source node. This latter has to push to the controller at the collection points on its own without the need for a request message.
- On the other hand, our proposed approach leverages the cumulative, non-negative, and non-decreasing aspects of the counters in predicting near-future

values based on the previously collected ones. We use time series forecasting (ARIMA and Double Exponential Smoothing).

COCO has the advantage of being lightweight for switches and easy to deploy since almost all the intelligence is concentrated on the collector in the controller. The evaluation results with real traces show that our proposed approach can reduce the number of pushed messages up to 75% compared to a fixed periodic data collection with a very good accuracy represented by a collection error of less than 0.5%.

ML-based Performance Modeling in SD-DCN (Chapter 5). For our second contribution, we tackle complex management operations in DCNs. DCN management operations required performance models, which handcrafted analytical modeling approaches struggle to provide due to the DCN environment dynamicity and complexity. Powering data-driven analytics, we propose a machine learning framework build upon SDN for incast performance prediction. With its capability of not relying on any domain-specific assumptions and approximations, machine learning can then be leveraged to construct a generalized model via a uniform training method. We carry out intensive experiments with the NS-3 simulator and build the needed dataset. Using this dataset, we designed machine learning incast completion time prediction models using random forest regression. The evaluation results show the effectiveness of our ML-modeling approach that can accurately capture complex relationships between various factors and incast completion time.

Automatic ML and BO based smart switch buffer management in SD-DCN (Chapter 6). Finally, for the third contribution, we leverage the ML-modeling approach from the previous contribution with BO to design an intelligent switch buffer management scheme in DCN. It provides great performance for mixed incast and elephant traffic, meaning maximal throughput for elephant traffic and minimal completion time for incast. Choosing the proper buffer size and AQM parameters for optimal performance is challenging due to the complexity and dynamics of the data center environment. More, these parameters interact in complex ways, and then it isn't easy to come up with analytical performance models that can guide the choices. We are then in the presence of a difficult optimization problem where the objective function (performance model) is unknown. Thanks to Bayesian Optimization, a powerful tool designed to solve such problems, where the objective has no simple closed-form. BO supposes, however, that the performance function can be evaluated at any parameter combinations. But this is not possible for online decision-making by the adaptive smart buffering scheme. We leverage the ML performance model build upon historical data of past parameter configurations and their observed performance to handle this issue. The ML model provides such evaluations through predictions. Coupled with SDN, this BO&ML-based buffer

management scheme can select the best or near-best parameter settings to deliver overall good performance to mixed incast and elephant traffic in DCNs. It uses dynamic automatic optimization and can adapt automatically to continuous traffic changes. At last, it is aligned with the self-driving networking trend that allows network systems to deal with dynamic workloads in an automated and, ideally, zero-touch way.

7.2 Future Directions

SDN Monitoring Framework and COCO

In a short term perspective, We will investigate the pertinence of our proposed approach on more traces, different type of traffic, i.e., traffic with different magnitude of the number of flows, type of flows TCP or UDP, and at the same time by improving our proposition according to these specific aspects. For the variety of the data to be collected, we will investigate COCO on other continuous statistics, either on switches or end hosts, that do not present an increasing trend, for instance, switch queue occupancy, CPU usage, etc. For such statistics where the time series may be more irregular, we can use Long Short-Term Memory (LSTM)/Recurrent Neural Network (RNN).

Although our proposed solution requires minimal user-defined parameters (only T_0 the data freshness period, and η the collection imprecision tolerance indicator), we can aspire for a more autonomous monitoring procedure that may adjust the accuracy-overhead tradeoff automatically based on a high-level goal defined by the management plane. With its ability to interact with a dynamic environment and learn optimal policies, reinforcement learning seems to be an excellent candidate to investigate from a mid-term perspective.

Finally, as a long-term perspective concerning SDN monitoring, we plan to rethink the proposed framework in-band network telemetry (INT coupled) with P4. With this feature, our framework will enable the monitoring of end-to-end information and then will not only cover the passive monitoring on data plane switches. It thus will integrated end hosts. This vision is in some sense aligned with the top-down approach for network telemetry discussed in [Yu 2019], where network monitoring systems should provide high-level declarative abstractions for network management teams to specifies measurement queries. This queries may be translated into low-level API calls at switches and hosts.

ML-based SD-DCN management

Secondly, for the ML-based DCN management propositions, the identified future works are as follow. As short-term future works, we plan to extend the dataset with new parameters. For instance, congestion control and queuing discipline schemes

(e.g., BBR) and by integrating new features in the dataset as the SRU, RTO_{min}, etc. Indeed, BBR, for example, is not yet part of the NS-3 simulator and was not considered when constructing our working datasets. For the ML&BO smart buffer management that focuses on TCP Cubic and FQ-Codel as the most promising candidate to handle mixed DCN workload and as a proof-of-concept of our proposition, we plan to investigate performance improvement using other transport protocols (e.g., DCTCP) with our proposed scheme. Also, a more general performance metric $U = \log(\gamma) - \alpha * \log(\tau)$ use needs to be investigated for the optimization process. This general objective may help to control the priority given of the incast compared to the elephant traffic.

As of mid-term perspectives, we plan to associate with the ML&BO optimal parameters finding Network Calculus [Le Boudec 2000] to determine worse case bounds of the DCN traffic performance. This will brings more flexibility in ensuring QoS when the optimal performance can not be achieved. This way, the degraded mode will be possible. Additionally, for the ML&BO loop, we will investigate how RL and Deep RL can be helpful, especially when dealing with a high number of features that may explode the possible network states to consider in the performance optimization process.

Final Remarks

Last but not least, more high-level research may try to determine the types of tasks that can really be full-automated, the ones that still need humans in the loop. For this latter how human and machine may collaborate to make the internet great again need to be questioned.

ML as an alternative or an assistant to human operations to cope with complexity is a promising approach. In the case of networking, since networks are significantly present in our daily life, they undergo a large amount of traffic, and the requirements are high. This resulted network system is then complex, and this complexity tends to magnify. ML can help, as we show it in this thesis. But, at this point, it's still difficult to say what problems ML can or cannot solve. However, if we cannot solve the problem, is it "normal" or is it because network-specific MlL algorithms are not yet developed. All these interesting research questions need further investigations and collaborations. These investigations may pass through multidisciplinary research[2], beginning by collaborating with the AI/ML community and broader areas.

[2] "Real problems are often interdisciplinary", Jennifer Rexford, Athena 2017

7.3 Publications

This thesis is founded on the knowledge acquired through the following scientific productions:

Conferences:

[1] *Kokouvi Benoit Nougnanke*, Yann Labit, and Marc Bruyere, "BO&ML-based Smart Buffer Management in Software-Defined Data Center Networks," under *Review Process*.

[2] *Kokouvi Benoit Nougnanke*, Yann Labit, and Marc Bruyere, "ML-Based Incast Performance Optimization in Software-Defined Data Centers," in *22nd International Conference on High-Performance Switching and Routing (IEEE HPSR 2021)*, 7-10 June 2021. Paris, France.

[3] *Kokouvi Benoit Nougnanke*, Yann Labit, Marc Bruyere, Simone Ferlin, and Ulrich Matchi Aïvodji, "Learning-based Incast Performance Inference in Software-Defined Data Centers," in *24th Conference on Innovation in Clouds, Internet and Networks (IEEE ICIN 2021)*, March 01-04, 2021. [**Best Paper Award**].

[4] *Kokouvi Benoit Nougnanke*, Marc Bruyere, and Yann Labit, "Low-Overhead Near-Real-Time Flow Statistics Collection in SDN," in *6th IEEE International Conference on Network Softwarization (NetSoft 2020)*, June 2020.

[5] *Kokouvi Benoit Nougnanke*, and Yann Labit, "Novel Adaptive Data Collection based on a Confidence Index in SDN," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*, 10-13 Jan. 2020.

Posters:

[1] *Kokouvi Benoit Nougnanke*, Marc Bruyere, and Yann Labit, "Low-Cost Near-Real-Time Counters Collection in SDN," in *2020 ACM SIGCOMM Symposium on SDN Research (SOSR)*, March 2020.

[2] *Kokouvi Benoit Nougnanke*, and Yann Labit, " Control Messages Optimization in SDN," in *SDN DAY 2018*, November 2018.

Best Paper:

Appendix A

Résumé des Travaux de thèse

A.1 Contexte et Motivations

La croissance exponentielle des services et des applications Internet, ainsi que l'augmentation massive du trafic, complexifient les réseaux informatiques. Ces derniers atteignent un point où les fonctions de management (ou gestion) traditionnelles, principalement régies par des opérations humaines, ne parviennent pas à maintenir le réseau opérationnel. Ce constat soulève la nécessité de recourir à de nouvelles approches de management pour les réseaux modernes ainsi que ceux du futur. Dans ce contexte, le Software Defined-Networking (SDN) émerge comme une nouvelle architecture pour la gestion des réseaux [Casado 2019, Feamster 2014]. Sa caractéristique principale est la séparation du plan de contrôle du plan de données des équipements réseaux. SDN propose une gestion centralisée et intelligente des réseaux à l'aide d'applications logicielles. Il rend les réseaux programmables en apportant de la flexibilité dans leur contrôle et leur management.

SDN présente, certes, des avantages (abstractions au niveau du plan de contrôle, interfaces de programmation ouvertes, etc.) mais il n'est pas exempt de nouveaux défis dont la construction de la vision globale du réseau par une collecte efficace de données. Cependant, il est porteur d'un avenir prometteur pour les réseaux. Ainsi l'adoption du SDN et l'identification des défis restants permettraient de tendre vers un mécanisme de management de réseau plus adapté. En d'autres termes, avec le SDN, le management réseau a atteint un niveau intéressant. Cependant, de nouvelles recherches doivent être menées pour que la gestion du réseau soit plus efficace et puisse supporter les réseaux modernes et futurs. Nous identifions ces deux challenges principaux:

Besoin de systèmes de monitoring efficaces avec SDN. A titre d'illustration, bien que la séparation du plan de contrôle du plan de données présente de nombreux avantages (flexibilité, programmabilité, interfaces ouvertes, etc.), les communications entre le plan de données et le contrôleur SDN génèrent une surcharge (overhead) importante [Curtis 2011]. Cette surcharge concerne d'une part le contrôle et la configuration par l'installation de règles de flux, la gestion des paquets entrants et sortants, et d'autre part la collecte d'informations sur le réseau. Il est donc important de concevoir des systèmes de monitoring (télémétrie) efficaces avec un bon compromis précision-overhead.

Besoin de plus d'automatisation pour les tâches complexes. L'autonomie accrue offerte par le SDN est appropriée mais pas suffisante au regard de la complexité croissante du réseau (cas d'étude des data centers) qui doit accueillir des

charges de travail hétérogènes avec des exigences de performance variées. En effet, même avec le SDN, le rôle de l'opérateur humain dans la gestion du réseau est très important, même pour les tâches liées à la prise de décision en "temps réel"/"quasi temps réel". Cette approche peut se traduire par un coût humain terriblement élevé [Clark 2016]. Et comme si cela ne suffisait pas, la plupart des pannes de réseau sont causées par des erreurs manuelles humaines. Il est alors nécessaire de rechercher de plus en plus d'autonomie dans les tâches de gestion, et l'intelligence artificielle (IA)/Machine Learning (ML) peut y contribuer.

Par conséquent, assurer de bonnes performances des réseaux avec les services et applications qui évoluent rapidement, même avec le SDN, devient un défi surtout en cause des boucles de contrôle classiques impliquant une forte intervention ou une dépendance humaine. C'est ainsi qu'inspirée par le concept des véhicules autonomes, la vision du self-driving network (SelfDN), s'appuie sur le SDN et le ML pour simplifier la gestion des réseaux et tendre idéalement vers des réseaux autonomes a fait l'objet d'une attention considérable [Feamster 2017, Kalmbach 2018]. Cette vision des réseaux autonomes, avec le potentiel de simplifier la gestion des réseaux, est un "grand challenge" pour les réseaux et le "Saint Graal" à atteindre pour la recherche sur le management des réseaux.

A.2 Nos Contributions

En phase avec la vision SelfDN et en prenant comme cas d'étude les réseaux de data centers (voir Figure A.1), nous avons proposé les trois contributions suivantes en adressant les challenges soulevés plus haut:

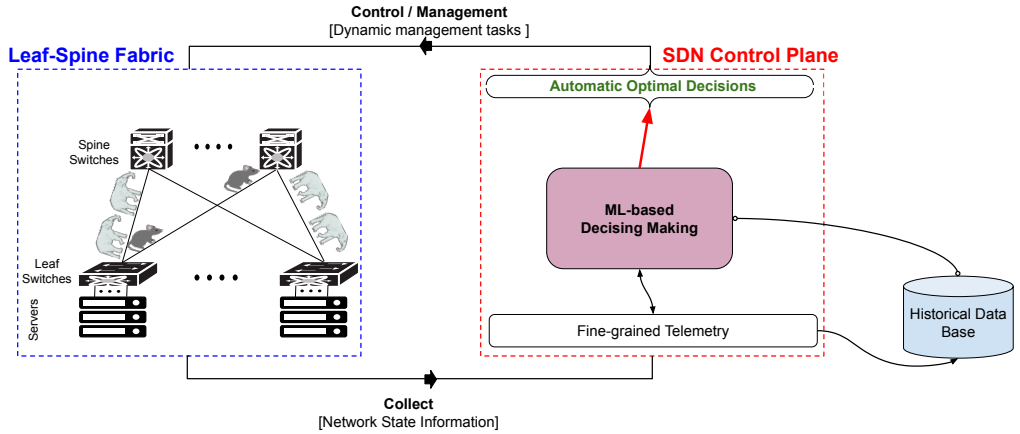

Figure A.1: Vers le SelfDN pour le management du SD-DCN (Réseaux data centers pilotés par SDN)

Monitoring SDN efficace (Chapitre 3). Dans notre première contribution,

nous proposons un framework de monitoring générique pour le SDN afin de fournir une collecte de données efficace pour construire la vue globale et la plus à jour du plan de données sous-jacent au niveau du contrôleur SDN. Ce framework distingue la collecte de données basée sur des événements, la collecte de données ad-hoc avec des messages requête/réponse en mode pull, et enfin, la collecte de statistiques périodiques (compteurs). Pour cette dernière collecte, nous sommes en présence d'un dilemme qui concerne la période de collecte optimale à utiliser pour obtenir un bon compromis précision-overhead. Contrairement aux travaux de la littérature qui proposent des mécanismes de collecte avec une fréquence adaptative, nous proposons une solution adaptative plus efficace appelée COCO (**CO**nfidence based adaptive **CO**llection). COCO est une approche de collecte de données périodique adaptative basée sur le modèle de collecte push et la prédiction de séries temporelles. Pour fournir une évolution précise des compteurs de flux (d'octets) avec un faible overhead, COCO saisit les deux opportunités d'optimisation suivantes:

- D'une part, le modèle pull avec des messages requête-réponse de la collecte OpenFlow [McKeown 2008] standard est inefficace. Avec une collecte périodique ou avec notre collecte adaptative, les points de collecte dans le temps sont connus par le nœud source. Ce dernier peut donc pousser vers le contrôleur aux points de collecte lui-même sans avoir recours à un message requête.
- D'autre part, COCO exploite l'aspect cumulatif et croissant des compteurs lors de la phase de prédiction des valeurs futures (de proche en proche) à partir des valeurs précédemment collectées. Nous utilisons des méthodes de prédiction des séries temporelles à l'instar d'ARIMA et du lissage exponentiel double.

COCO a l'avantage d'être léger pour les commutateurs et facile à déployer puisque presque toute l'intelligence et les calculs lourds sont concentrés sur le collecteur dans le contrôleur SDN. Les résultats de l'évaluation avec des traces réelles montrent que l'approche proposée peut réduire le nombre de messages poussés jusqu'à 75 % par rapport à une collecte de données périodique fixe, avec une très bonne précision représentée par une erreur de collecte inférieure à 0,5 %.

Modéles de performance basés ML dans SD-DCN (Chapitre 5). Dans notre deuxième contribution, nous nous attaquons aux opérations de gestion complexes des réseaux DCN. Les opérations de gestion des DCN nécessitent des modèles de performance, que les approches de modélisation analytique classiques peinent à fournir en raison de la dynamique et de la complexité de l'environnement DCN. En utilisant des analyses basées sur les données, nous proposons un framework d'apprentissage automatique au-dessus de SDN pour la prédiction des performances incast (trafic de type N-à-1 où plusieurs serveurs communiquent avec un client simultanément) [Handley 2017]. Grâce à sa capacité à ne pas s'appuyer sur des

hypothèses et des approximations simplificatrices ni de connaissances "domain-specific", le ML est utilisé pour construire un modèle généralisable facilement. La construction du modèle ML est faite en suivant le workflow classique [Wang 2017] d'application du ML aux réseaux. Nous avons réalisé des expériences intensives avec le simulateur NS-3 et construit le dataset nécessaire. À l'aide de ce dataset, nous avons conçu le modèle ML du temps de finition du trafic incast en utilisant la régression par forêt aléatoire (Random Forest). Les résultats de l'évaluation montrent l'efficacité de notre approche de modélisation ML qui peut capturer avec précision les relations complexes entre les différents paramètres et le temps de finition de l'incast.

Gestion automatique et intelligente des commutateurs (Smart buffering) basée sur ML et BO dans SD-DCN (Chapitre 6). Enfin, dans la troisième contribution, nous tirons parti de l'approche de modélisation ML de la contribution précédente avec l'optimisation bayésienne (BO) [Shahriari 2015] pour concevoir une solution de gestion intelligente des commutateurs dans l'environnement DCN. Elle offre de bonnes performances pour le trafic mixte incast et elephant, c'est-à-dire un débit maximal pour le trafic éléphant et un temps de finition minimal pour l'incast. Le choix de la taille de buffer et des paramètres AQM appropriés pour une performance optimale est challengeant en raison de la complexité et de la dynamique de l'environnement des data centers. De plus, ces paramètres interagissent de manière complexe, et il n'est donc pas facile de trouver des modèles analytiques de performance qui puissent guider les choix. Nous sommes alors en présence d'un problème d'optimisation difficile où la fonction objectif (modèle de performance) est inconnue. Grâce à l'optimisation bayésienne, un outil puissant conçu pour résoudre de tels problèmes, où l'objectif n'a pas de forme analytique simple. L'optimisation bayésienne suppose toutefois que la fonction de performance puisse être évaluée pour n'importe quelle combinaison de paramètres. Mais cela n'est pas possible pour la prise de décision en mode on-line par la solution de smart buffering des commutateurs. Nous utilisons le modèle de performance ML basé sur les données historiques des configurations de paramètres passées et leurs performances observées pour résoudre ce problème. Le modèle ML fournit de telles évaluations par le biais de prédictions. Couplé au SDN, cette solution de gestion intelligente des commutateurs basée sur BO&ML peut sélectionner les meilleurs paramètres afin de fournir une bonne performance globale au trafic mixte incast et éléphant dans les DCN. La solution utilise une optimisation dynamique automatique et peut s'adapter automatiquement aux changements continus du trafic. Enfin, elle s'aligne sur la tendance des réseaux autonomes avec pour vision de permettre aux systèmes réseaux de gérer les charges de travail dynamiques de manière automatisée et, idéalement, sans intervention humaine.

La vue d'ensemble de la thèse telle qu'illustrée dans Figure A.2 met en évidence notre vision de l'Internet et les défis concernés et adressés par cette thèse.

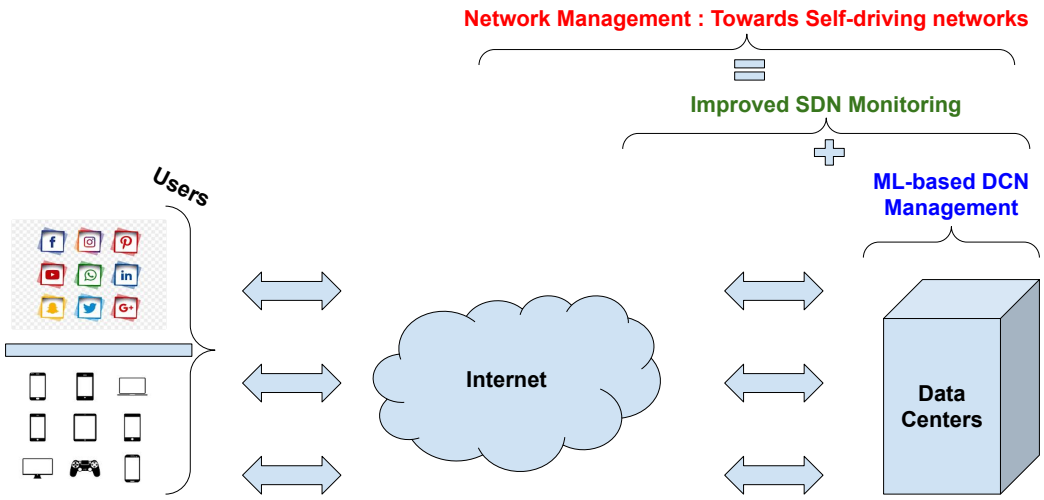

Figure A.2: Vue d'ensemble de la thèse

Remarques Finales

Finalement, nous tenons à souligner que l'approche ML défendue dans cette thèse n'implique pas de sortir l'homme complètement de la boucle. Enfin et surtout, des recherches de plus haut niveau pourraient tenter de déterminer les types de tâches qui peuvent réellement être entièrement automatisées, celles qui nécessitent toujours la présence de l'homme dans la boucle. Il conviendra de s'interroger sur la manière dont l'opérateur humain et le ML peuvent collaborer pour rendre l'Internet "Great Again".

Le ML en tant qu'alternative ou assistant aux opérations humaines pour faire face à la complexité grandissante des réseaux est une approche prometteuse. Dans le cas des réseaux, étant donné que les réseaux sont très présents dans notre vie quotidienne, ils subissent une grande quantité de trafic et des exigences élevées en termes de performance. Le système de réseau qui en résulte est alors complexe, et cette complexité a tendance à s'amplifier. Le ML peut aider, comme nous le montrons dans cette thèse. Mais, à ce stade, il est encore difficile de dire quels problèmes le ML peut ou ne peut pas résoudre. Cependant, si nous ne pouvons pas résoudre le problème, est-ce "normal" ou est-ce parce que les algorithmes ML spécifiques au réseau ne sont pas encore développés. Toutes ces questions de recherche intéressantes nécessitent des investigations et des collaborations supplémentaires. Ces investigations peuvent passer par une recherche multidisciplinaire[1], en commençant par une consolidation des collaborations avec la communauté AI/ML ainsi que des domaines plus larges.

[1] "Real problems are often interdisciplinary", Jennifer Rexford, Athena 2017

Appendix B

Machine Learning & Bayesian Optimization

B.1 Choosing the right ML estimator

Choosing the right ML algorithm is not the funny part of ML. There is no theory yet, and the choice is much about art. However, different estimators are better suited for different types of data and different problems. Figure B.1 from `https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html` provides users a bit of a rough guide on which algorithms to try for a specific problem. The map on the website is interactive and provides documentation for each algorithm.

B.2 BO Procedure

Figure B.2 provides an illustration of the BO procedure for a one variable function. The plots show the mean and confidence intervals estimated with a probabilistic model of the objective function. Although the objective function is shown, in practice, it is unknown. The plots also show the acquisition functions in the lower shaded plots. The acquisition is high where the model predicts a high objective (exploitation) and where the prediction uncertainty is high (exploration). Note that the area on the far left remains unsampled, as while it has high uncertainty, it is correctly predicted to offer little improvement over the highest observation

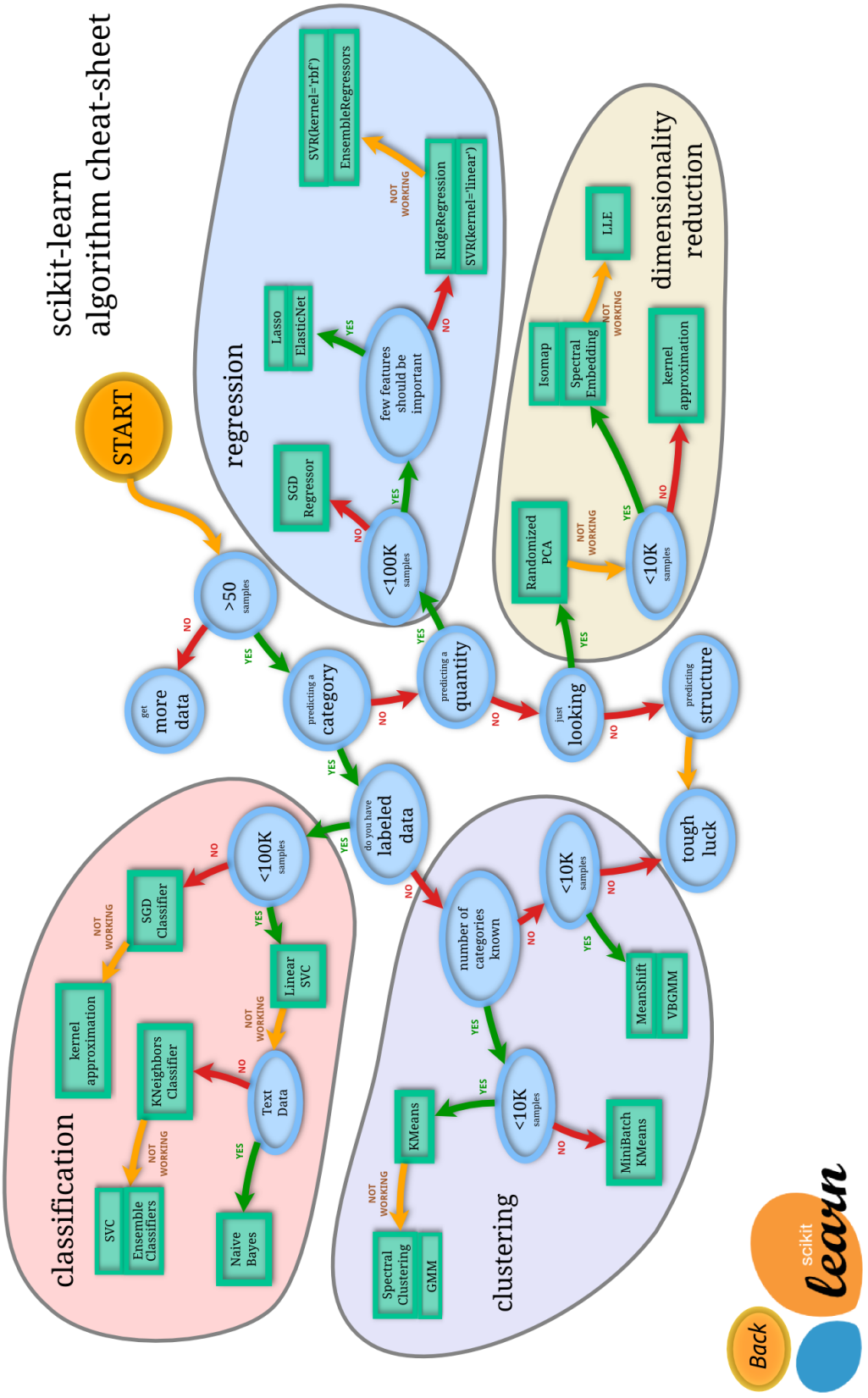

Figure B.1: Scikit-learn ML MAP

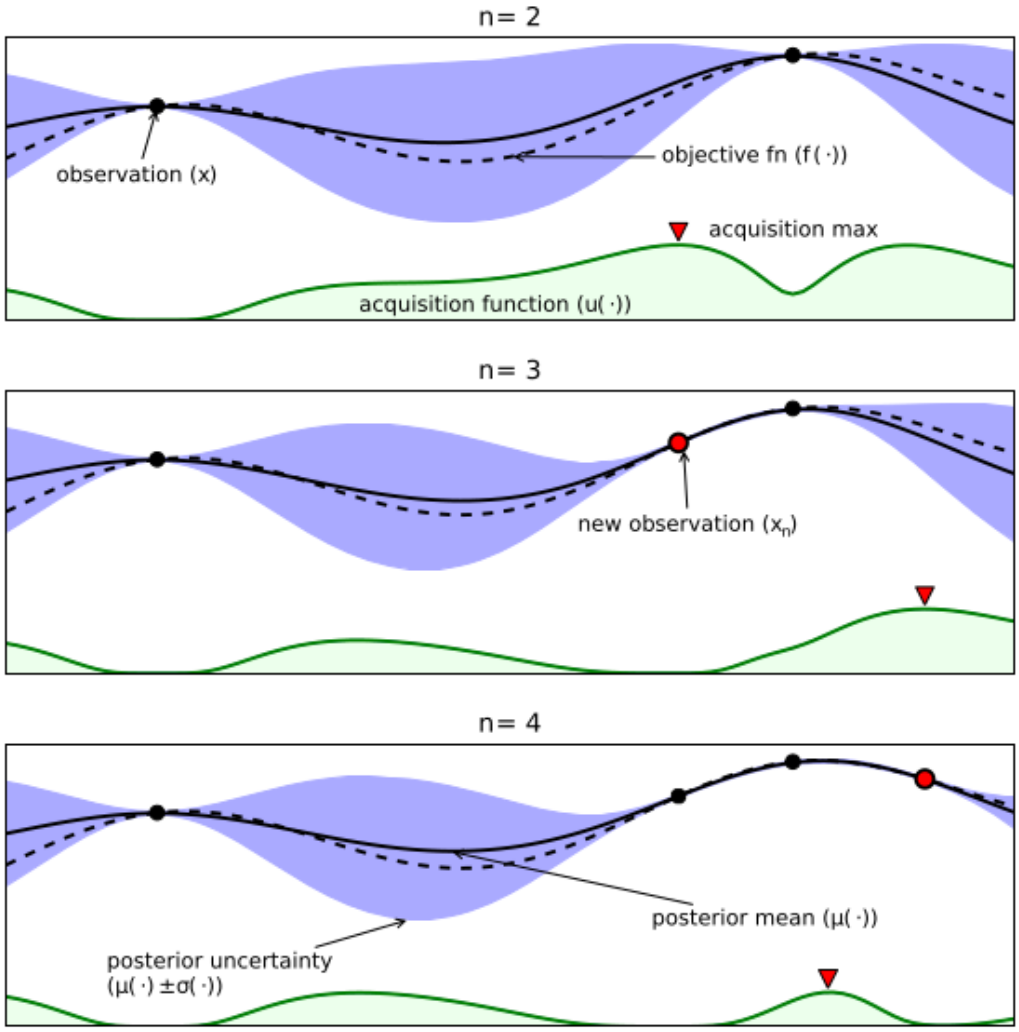

Figure B.2: Illustration of the Bayesian optimization procedure over three iterations (From [Shahriari 2015])

Bibliography

[Akin 2019] Erdal Akin and Turgay Korkmaz. *Comparison of Routing Algorithms With Static and Dynamic Link Cost in Software Defined Networking (SDN).* IEEE Access, vol. 7, pages 148629–148644, 2019. (Cited in page 30.)

[Alipourfard 2017] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu and Ming Zhang. *Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics.* In 14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17), pages 469–482, 2017. (Cited in pages 55 and 83.)

[Alizadeh 2010a] Mohammad Alizadeh, Albert Greenberg, Dave Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta and Murari Sridharan. *DCTCP: Efficient packet transport for the commoditized data center.* 2010. (Cited in pages 54 and 55.)

[Alizadeh 2010b] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta and Murari Sridharan. *Data center tcp (dctcp).* In Proceedings of the ACM SIGCOMM 2010 conference, pages 63–74, 2010. (Cited in pages 64 and 82.)

[Alizadeh 2013] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar and Scott Shenker. *pfabric: Minimal near-optimal datacenter transport.* ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pages 435–446, 2013. (Cited in pages 54 and 56.)

[Alizadeh 2014] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav*et al. CONGA: Distributed congestion-aware load balancing for datacenters.* In Proceedings of the 2014 ACM Conference on SIGCOMM, pages 503–514, 2014. (Cited in page 54.)

[Alom 2015] Md Zahangir Alom, VenkataRamesh Bontupalli and Tarek M Taha. *Intrusion detection using deep belief networks.* In 2015 National Aerospace and Electronics Conference (NAECON), pages 339–344. IEEE, 2015. (Cited in page 59.)

[Amaral 2016] Pedro Amaral, Joao Dinis, Paulo Pinto, Luis Bernardo, Joao Tavares and Henrique S Mamede. *Machine learning in software defined networks: Data collection and traffic classification.* In 2016 IEEE 24th International conference on network protocols (ICNP), pages 1–5. IEEE, 2016. (Cited in page 59.)

[Appenzeller 2004] Guido Appenzeller, Isaac Keslassy and Nick McKeown. *Sizing router buffers*. ACM SIGCOMM Computer Communication Review, vol. 34, no. 4, pages 281–292, 2004. (Cited in page 82.)

[Ayoubi 2018] Sara Ayoubi, Noura Limam, Mohammad A Salahuddin, Nashid Shahriar, Raouf Boutaba, Felipe Estrada-Solano and Oscar M Caicedo. *Machine learning for cognitive network management*. IEEE Communications Magazine, vol. 56, no. 1, pages 158–165, 2018. (Cited in pages ix, 7, and 8.)

[Azzouni 2017] Abdelhadi Azzouni, Raouf Boutaba and Guy Pujolle. *NeuRoute: Predictive dynamic routing for software-defined networks*. In 2017 13th International Conference on Network and Service Management (CNSM), pages 1–6. IEEE, 2017. (Cited in page 59.)

[Benson 2010] Theophilus Benson, Aditya Akella and David A Maltz. *Network traffic characteristics of data centers in the wild*. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, pages 267–280. ACM, 2010. (Cited in page 38.)

[Bonfoh 2018] El-Fadel Bonfoh, Samir Medjiah and Christophe Chassot. *A Parsimonious Monitoring Approach for Link Bandwidth Estimation within SDN-based Networks*. In 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), pages 512–516. IEEE, 2018. (Cited in page 44.)

[Bosshart 2014] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese*et al. P4: Programming protocol-independent packet processors*. ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pages 87–95, 2014. (Cited in page 44.)

[Boutaba 2018] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano and Oscar M Caicedo. *A comprehensive survey on machine learning for networking: evolution, applications and research opportunities*. Journal of Internet Services and Applications, vol. 9, no. 1, page 16, 2018. (Cited in pages 24, 58, and 60.)

[Bufferbloat 2014] Bufferbloat. *Best Practices for Benchmarking CoDel and FQ CoDel (and almost any other network subsystem!)*. `https://www.bufferbloat.net/projects/codel/wiki/Best_practices_for_benchmarking_Codel_and_FQ_Codel/`, 2014. (Cited in pages 79 and 82.)

[Candela 2020] Massimo Candela, Valerio Luconi and Alessio Vecchio. *Impact of the COVID-19 pandemic on the Internet latency: A large-scale study*. Computer Networks, vol. 182, page 107495, 2020. (Cited in page 1.)

[Casado 2019] Martín Casado, Nick McKeown and Scott Shenker. *From ethane to SDN and beyond*. ACM SIGCOMM Computer Communication Review, vol. 49, no. 5, pages 92–95, 2019. (Cited in pages 4, 20, 44, and 101.)

[Castanheira 2019] Lucas Castanheira, Ricardo Parizotto and Alberto E Schaeffer-Filho. *Flowstalker: Comprehensive traffic flow monitoring on the data plane using p4.* In ICC 2019-2019 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2019. (Cited in page 45.)

[Cerf 1974] Vinton Cerf and Robert Kahn. *A protocol for packet network intercommunication.* IEEE Transactions on communications, vol. 22, no. 5, pages 637–648, 1974. (Cited in page 2.)

[Chen 2009] Yanpei Chen, Rean Griffith, Junda Liu, Randy H Katz and Anthony D Joseph. *Understanding TCP incast throughput collapse in datacenter networks.* In Proceedings of the 1st ACM workshop on Research on enterprise networking, pages 73–82, 2009. (Cited in pages 56, 64, and 74.)

[Chen 2015] Wen Chen, Fengyuan Ren, Jing Xie, Chuang Lin, Kevin Yin and Fred Baker. *Comprehensive understanding of TCP Incast problem.* In 2015 IEEE Conference on Computer Communications (INFOCOM), pages 1688–1696. IEEE, 2015. (Cited in pages 56, 64, and 74.)

[Chen 2016a] Li Chen, Kai Chen, Wei Bai and Mohammad Alizadeh. *Scheduling mix-flows in commodity datacenters with karuna.* In Proceedings of the 2016 ACM SIGCOMM Conference, pages 174–187, 2016. (Cited in page 54.)

[Chen 2016b] Zhitang Chen, Jiayao Wen and Yanhui Geng. *Predicting future traffic using hidden markov models.* In 2016 IEEE 24th international conference on network protocols (ICNP), pages 1–6. IEEE, 2016. (Cited in page 58.)

[Chen 2018] Li Chen, Justinas Lingys, Kai Chen and Feng Liu. *Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization.* In Proceedings of the 2018 conference of the ACM special interest group on data communication, pages 191–205, 2018. (Cited in pages 54 and 93.)

[Cheng 2017] Guang Cheng and Jun Yu. *Adaptive sampling for openflow network measurement methods.* In Proceedings of the 12th International Conference on Future Internet Technologies, page 4. ACM, 2017. (Cited in page 45.)

[Chowdhury 2014] Shihabur Rahman Chowdhury, Md Faizul Bari, Reaz Ahmed and Raouf Boutaba. *Payless: A low cost network monitoring framework for software defined networks.* In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–9. IEEE, 2014. (Cited in pages 28, 30, 45, and 46.)

[Chuprikov 2020] Pavel Chuprikov, Sergey Nikolenko and Kirill Kogan. *Towards declarative self-adapting buffer management.* ACM SIGCOMM Computer Communication Review, vol. 50, no. 3, pages 30–37, 2020. (Cited in pages 55 and 82.)

[cis 2017] *Intelligent Buffer Management on Cisco Nexus 9000 Series Switches.* https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.pdf, 2017. (Cited in pages 54, 55, and 56.)

[Clark 1988] David Clark. *The design philosophy of the DARPA Internet protocols.* In Symposium proceedings on Communications architectures and protocols, pages 106–114, 1988. (Cited in page 2.)

[Clark 2003] David D Clark, Craig Partridge, J Christopher Ramming and John T Wroclawski. *A knowledge plane for the internet.* In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pages 3–10, 2003. (Cited in page 9.)

[Clark 2016] David Clark, Jennifer Rexford and Amin Vahdat. *A purpose-built global network: Google's move to sdn.* Communications of the ACM, vol. 59, no. 3, pages 46–54, 2016. (Cited in pages 54, 60, and 102.)

[Crankshaw 2019] Daniel Crankshaw. *The Design and Implementation of Low-Latency Prediction Serving Systems.* PhD thesis, UC Berkeley, 2019. (Cited in page 74.)

[Curtis 2011] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma and Sujata Banerjee. *DevoFlow: Scaling flow management for high-performance networks.* In ACM SIGCOMM Computer Communication Review, volume 41, pages 254–265. ACM, 2011. (Cited in pages 30 and 101.)

[da Silva 2016] Anderson Santos da Silva, Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville and Alberto Schaeffer-Filho. *ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN.* In NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, pages 27–35. IEEE, 2016. (Cited in page 60.)

[Dewancker 2015] Ian Dewancker, Michael McCourt and Scott Clark. *Bayesian optimization primer.* URL https://app. sigopt. com/static/pdf/SigOpt_Bayesian_Optimization_Primer. pdf, 2015. (Cited in page 84.)

[Edmund 1993] S Yu Edmund and CY Roger Chen. *Traffic prediction using neural networks.* In Proceeding of Globecom, volume 2, pages 991–995, 1993. (Cited in page 58.)

[ETSI-ZSM 2019] ETSI-ZSM. *ZSM Reference Architecture.* https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf, 2019. (Cited in page 6.)

[Fan 2017] Zhong Fan and Ran Liu. *Investigation of machine learning based network traffic classification.* In 2017 International Symposium on Wireless Communication Systems (ISWCS), pages 1–6. IEEE, 2017. (Cited in page 59.)

[Fawcett 2018] Lyndon Fawcett, Sandra Scott-Hayward, Matthew Broadbent, Andrew Wright and Nicholas Race. *TENNISON: A Distributed SDN Framework for Scalable Network Security.* IEEE Journal on Selected Areas in Communications, vol. 36, no. 12, pages 2805–2818, 2018. (Cited in page 46.)

[Feamster 2005] Nick Feamster and Hari Balakrishnan. *Detecting BGP configuration faults with static analysis.* In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2, pages 43–56, 2005. (Cited in page 19.)

[Feamster 2014] Nick Feamster, Jennifer Rexford and Ellen Zegura. *The road to SDN: an intellectual history of programmable networks.* ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pages 87–98, 2014. (Cited in pages 4, 20, 65, and 101.)

[Feamster 2017] Nick Feamster and Jennifer Rexford. *Why (and how) networks should run themselves.* arXiv preprint arXiv:1710.11583, 2017. (Cited in pages 6, 7, and 102.)

[Feldmann 2001] Anja Feldmann and Jennifer Rexford. *IP network configuration for intradomain traffic engineering.* IEEE Network, vol. 15, no. 5, pages 46–57, 2001. (Cited in page 19.)

[Floyd 2008] Sally Floyd*et al. Metrics for the evaluation of congestion control mechanisms.* Technical Report, RFC 5166, March, 2008. (Cited in page 79.)

[Foster 2020] Nate Foster, Nick McKeown, Jennifer Rexford, Guru Parulkar, Larry Peterson and Oguz Sunay. *Using deep programmability to put network owners in control.* ACM SIGCOMM Computer Communication Review, vol. 50, no. 4, pages 82–88, 2020. (Cited in page 65.)

[Found 2015] Open Networking Found. *Openflow switch specification version 1.5. 1 (Protocol version 0x06)*, 2015. (Cited in pages ix and 22.)

[Fu 2021] Silvery Fu, Saurabh Gupta, Radhika Mittal and Sylvia Ratnasamy. *On the Use of ML for Blackbox System Performance Prediction.* In 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). USENIX Association, April 2021. (Cited in pages 81 and 93.)

[Gao 2018] Kai Gao, Jingxuan Zhang, Y Richard Yang and Jun Bi. *Prophet: Fast accurate model-based throughput prediction for reactive flow in DC networks.* In IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pages 720–728. IEEE, 2018. (Cited in page 59.)

[Garcia-Teodoro 2009] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández and Enrique Vázquez. *Anomaly-based network intrusion detection: Techniques, systems and challenges.* computers & security, vol. 28, no. 1-2, pages 18–28, 2009. (Cited in page 59.)

[Geurts 2004] Pierre Geurts, Ibtissam El Khayat and Guy Leduc. *A machine learning approach to improve congestion control over wireless computer networks.* In Fourth IEEE International Conference on Data Mining (ICDM'04), pages 383–386. IEEE, 2004. (Cited in page 58.)

[Gomez 2019] Cesar A Gomez, Xianbin Wang and Abdallah Shami. *Intelligent active queue management using explicit congestion notification.* In 2019 IEEE Global Communications Conference (GLOBECOM), pages 1–6. IEEE, 2019. (Cited in page 55.)

[Gong 2018] YiXi Gong, James Roberts and Dario Rossi. *Per-Flow Fairness in the Datacenter Network.* In 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), pages 1–6. IEEE, 2018. (Cited in page 79.)

[Haleplidis 2015] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, J Hadi Salim, David Meyer and Odysseas Koufopavlou. *Software-defined networking (SDN): Layers and architecture terminology.* Technical Report, 2015. (Cited in pages 4 and 20.)

[Hamza 2019] Ayyoob Hamza, Hassan Habibi Gharakheili, Theophilus A Benson and Vijay Sivaraman. *Detecting Volumetric Attacks on loT Devices via SDN-Based Monitoring of MUD Activity.* In Proceedings of the 2019 ACM Symposium on SDN Research, pages 36–48. ACM, 2019. (Cited in page 31.)

[Handley 2017] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi and Marcin Wójcik. *Re-architecting datacenter networks and stacks for low latency and high performance.* In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pages 29–42, 2017. (Cited in pages 55, 56, and 103.)

[Harrison 2018] Rob Harrison, Qizhe Cai, Arpit Gupta and Jennifer Rexford. *Network-wide heavy hitter detection with commodity switches.* In Proceedings of the Symposium on SDN Research, page 8. ACM, 2018. (Cited in pages 31 and 44.)

[Hartung 2017] Marc Hartung and Marc Körner. *SOFTmon-Traffic Monitoring for SDN.* Procedia Computer Science, vol. 110, pages 516–523, 2017. (Cited in page 28.)

[Hassidim 2013] Avinatan Hassidim, Danny Raz, Michal Segalov and Ariel Shaqed. *Network utilization: The flow view.* In 2013 Proceedings IEEE INFOCOM, pages 1429–1437. IEEE, 2013. (Cited in page 42.)

[Hoeiland-Joergensen 2018] T Hoeiland-Joergensen, P McKenney, D Taht and J Gettys. *E. Dumazet," The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm.* Technical Report, RFC 8290, DOI 10.17487/RFC8290, January 2018, 2018. (Cited in pages 78 and 82.)

[Horn 2001] Paul Horn. *Autonomic computing: IBM's perspective on the state of information technology.* 2001. (Cited in page 7.)

[Hu 2010] Tiansi Hu and Yunsi Fei. *QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks.* IEEE Transactions on Mobile Computing, vol. 9, no. 6, pages 796–809, 2010. (Cited in page 58.)

[Hu 2020] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan and Yi Wang. *Aeolus: A building block for proactive transport in datacenters.* In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pages 422–434, 2020. (Cited in page 55.)

[Huang 2018] Xinli Huang, Shang Cheng, Kun Cao, Peijin Cong, Tongquan Wei and Shiyan Hu. *A survey of deployment solutions and optimization strategies for hybrid SDN networks.* IEEE Communications Surveys & Tutorials, vol. 21, no. 2, pages 1483–1507, 2018. (Cited in page 60.)

[Huawei 2020] Huawei. *ADN Solution White Paper (Autonomous Driving Network).* `https://carrier.huawei.com/~/media/CNBGV2/download/adn/Autonomous-Driving-Network-whitepaper-en1.pdf`, 2020. (Cited in page 6.)

[hus 2019] *Sizing the buffer.* `https://blog.apnic.net/2019/12/12/sizing-the-buffer/`, 2019. (Cited in page 82.)

[Hyndman 2006] Rob J Hyndman and Anne B Koehler. *Another look at measures of forecast accuracy.* International journal of forecasting, vol. 22, no. 4, pages 679–688, 2006. (Cited in page 35.)

[Jain 2013] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu*et al.* *B4: Experience with a globally-deployed software defined WAN.* ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pages 3–14, 2013. (Cited in page 54.)

[Jain 2016] Shashwat Jain, Manish Khandelwal, Ashutosh Katkar and Joseph Nygate. *Applying big data technologies to manage QoS in an SDN.* In 2016 12th International Conference on Network and Service Management (CNSM), pages 302–306. IEEE, 2016. (Cited in pages 59 and 75.)

[Jiang 2016] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica and Hui Zhang. {*CFA*}*: A practical prediction system for video qoe optimization.* In 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), pages 137–150, 2016. (Cited in page 58.)

[Kalkan 2018] Kübra Kalkan, Levent Altay, Gürkan Gür and Fatih Alagöz. *JESS: Joint entropy-based DDoS defense scheme in SDN.* IEEE Journal on Selected Areas in Communications, vol. 36, no. 10, pages 2358–2372, 2018. (Cited in page 60.)

[Kalmbach 2018] Patrick Kalmbach, Johannes Zerwas, Peter Babarczi, Andreas Blenk, Wolfgang Kellerer and Stefan Schmid. *Empowering self-driving networks.* In Proceedings of the Afternoon Workshop on Self-Driving Networks, pages 8–14. ACM, 2018. (Cited in pages 6 and 102.)

[Katta 2016] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman and Jennifer Rexford. *Hula: Scalable load balancing using programmable data planes.* In Proceedings of the Symposium on SDN Research, pages 1–12, 2016. (Cited in page 54.)

[Kellerer 2019] Wolfgang Kellerer, Patrick Kalmbach, Andreas Blenk, Arsany Basta, Martin Reisslein and Stefan Schmid. *Adaptable and data-driven softwarized networks: Review, opportunities, and challenges.* Proceedings of the IEEE, vol. 107, no. 4, pages 711–731, 2019. (Cited in page 60.)

[Kephart 2003] Jeffrey O Kephart and David M Chess. *The vision of autonomic computing.* Computer, vol. 36, no. 1, pages 41–50, 2003. (Cited in page 7.)

[Kim 2008] Hyunchul Kim, Kimberly C Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos and KiYoung Lee. *Internet traffic classification demystified: myths, caveats, and the best practices.* In Proceedings of the 2008 ACM CoNEXT conference, pages 1–12, 2008. (Cited in page 58.)

[Kim 2015] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit and Lawrence J Wobker. *In-band network telemetry via programmable dataplanes.* In ACM SIGCOMM, 2015. (Cited in page 44.)

[Király 2020] Orsolya Király, Marc N Potenza, Dan J Stein, Daniel L King, David C Hodgins, John B Saunders, Mark D Griffiths, Biljana Gjoneska, Joël Billieux, Matthias Brand*et al. Preventing problematic internet use during the COVID-19 pandemic: Consensus guidance.* Comprehensive Psychiatry, page 152180, 2020. (Cited in page 1.)

[Kuklinski 2014] Slawomir Kuklinski and Prosper Chemouil. *Network management challenges in software-defined networks.* IEICE Transactions on Communications, vol. 97, no. 1, pages 2–9, 2014. (Cited in page 2.)

[Kumar 2020] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan*et al. Swift: Delay is simple and effective for congestion control in the datacenter.* In Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pages 514–528, 2020. (Cited in pages 54 and 55.)

[Labit 2014] Yann Labit. *Contributions transversales pour la mesure et le contrôle intelligent appliqués aux systèmes de communication*, 2014. HDR. (Cited in page 18.)

[Laiymani 2013] David Laiymani and Abdallah Makhoul. *Adaptive data collection approach for periodic sensor networks.* In 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pages 1448–1453. IEEE, 2013. (Cited in page 45.)

[Le Boudec 2000] J-Y Le Boudec and Patrick Thiran. *A short tutorial on network calculus. I. Fundamental bounds in communication networks.* In 2000 IEEE International Symposium on Circuits and Systems (ISCAS), volume 4, pages 93–96. IEEE, 2000. (Cited in page 99.)

[Lee 2014] Sihyung Lee, Kyriaki Levanti and Hyong S Kim. *Network monitoring: Present and future.* Computer Networks, vol. 65, pages 84–98, 2014. (Cited in page 18.)

[Li 2016] Yi Li, Hong Liu, Wenjun Yang, Dianming Hu and Wei Xu. *Inter-datacenter network traffic prediction with elephant flows.* In NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium, pages 206–213. IEEE, 2016. (Cited in page 58.)

[Li 2019] Yuliang Li, Rui Miao, Mohammad Alizadeh and Minlan Yu. {*DETER*}*: Deterministic* {*TCP*} *Replay for Performance Diagnosis.* In 16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19), pages 437–452, 2019. (Cited in page 56.)

[Mahajan 2002] Ratul Mahajan, David Wetherall and Tom Anderson. *Understanding BGP misconfiguration.* ACM SIGCOMM Computer Communication Review, vol. 32, no. 4, pages 3–16, 2002. (Cited in page 19.)

[Mao 2016] Hongzi Mao, Mohammad Alizadeh, Ishai Menache and Srikanth Kandula. *Resource management with deep reinforcement learning.* In Proceedings of the 15th ACM workshop on hot topics in networks, pages 50–56, 2016. (Cited in page 59.)

[Mao 2017] Bomin Mao, Zubair Md Fadlullah, Fengxiao Tang, Nei Kato, Osamu Akashi, Takeru Inoue and Kimihiro Mizutani. *A tensor based deep learning technique for intelligent packet routing.* In GLOBECOM 2017-2017

IEEE Global Communications Conference, pages 1–6. IEEE, 2017. (Cited in page 58.)

[Martin 2018] Angel Martin, Jon Egaña, Julián Flórez, Jon Montalbán, Igor G Olaizola, Marco Quartulli, Roberto Viola and Mikel Zorrilla. *Network resource allocation system for QoE-aware delivery of media services in 5G networks.* IEEE Transactions on Broadcasting, vol. 64, no. 2, pages 561–574, 2018. (Cited in page 59.)

[Mazel 2015] Johan Mazel, Pedro Casas, Romain Fontugne, Kensuke Fukuda and Philippe Owezarski. *Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection.* International Journal of Network Management, vol. 25, no. 5, pages 283–305, 2015. (Cited in page 59.)

[McKeown 2008] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner. *OpenFlow: enabling innovation in campus networks.* ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pages 69–74, 2008. (Cited in pages 22 and 103.)

[Mestres 2017] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett*et al. Knowledge-defined networking.* ACM SIGCOMM Computer Communication Review, vol. 47, no. 3, pages 2–10, 2017. (Cited in page 7.)

[Mijumbi 2014] Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, Maxim Claeys, Filip De Turck and Steven Latré. *Design and evaluation of learning algorithms for dynamic resource management in virtual networks.* In 2014 IEEE network operations and management symposium (NOMS), pages 1–9. IEEE, 2014. (Cited in page 59.)

[Mukkamala 2002] Srinivas Mukkamala, Guadalupe Janoski and Andrew Sung. *Intrusion detection using neural networks and support vector machines.* In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290), volume 2, pages 1702–1707. IEEE, 2002. (Cited in page 59.)

[Mushtaq 2012] M Sajid Mushtaq, Brice Augustin and Abdelhamid Mellouk. *Empirical study based on machine learning approach to assess the QoS/QoE correlation.* In 2012 17th European Conference on Networks and Optical Communications, pages 1–7. IEEE, 2012. (Cited in page 58.)

[ONF TR-502] ONF TR-502. *SDN architecture, Issue 1, June 2014.* (Cited in page 23.)

[ONF TR-516] ONF TR-516. *Framework for SDN: Scope and Requirements, Version 1.0, June 2015.* (Cited in pages 23 and 24.)

[ONF TR-521] ONF TR-521. *SDN Architecture, Issue 1.1, 2016.* (Cited in page 23.)

[Owezarski 2010] Philippe Owezarski, Johan Mazel and Yann Labit. *0day anomaly detection made possible thanks to machine learning.* In International Conference on Wired/Wireless Internet Communications, pages 327–338. Springer, 2010. (Cited in page 59.)

[Pasca 2017] S Thomas Valerrian Pasca, Siva Sairam Prasad Kodali and Kotaro Kataoka. *AMPS: Application aware multipath flow routing using machine learning in SDN.* In 2017 Twenty-third National Conference on Communications (NCC), pages 1–6. IEEE, 2017. (Cited in page 59.)

[Pasquini 2017] Rafael Pasquini and Rolf Stadler. *Learning end-to-end application QoS from OpenFlow switch statistics.* In 2017 IEEE Conference on Network Softwarization (NetSoft), pages 1–9. IEEE, 2017. (Cited in pages 59 and 75.)

[Pedregosa 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, vol. 12, pages 2825–2830, 2011. (Cited in page 68.)

[Phanishayee 2008] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson and Srinivasan Seshan. *Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems.* In FAST, volume 8, pages 1–14, 2008. (Cited in pages 55 and 64.)

[Rafique 2018] Danish Rafique and Luis Velasco. *Machine learning for network automation: Overview, architecture, and applications [invited tutorial].* Journal of Optical Communications and Networking, vol. 10, no. 10, pages D126–D143, 2018. (Cited in page 24.)

[Raiciu 2019] Costin Raiciu and Gianni Antichi. *NDP: rethinking datacenter networks and stacks two years after.* ACM SIGCOMM Computer Communication Review, vol. 49, no. 5, pages 112–114, 2019. (Cited in page 82.)

[Ridwan 2021] MA Ridwan, NAM Radzi, F Abdullah and YE Jalil. *Applications of Machine Learning in Networking: A Survey of Current Issues and Future Challenges.* IEEE Access, 2021. (Cited in page 58.)

[Rusek 2019] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros and Albert Cabellos-Aparicio. *Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN.* In Proceedings of the 2019 ACM Symposium on SDN Research, pages 140–151, 2019. (Cited in page 56.)

[Rusek 2020] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros and Albert Cabellos-Aparicio. *RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN.* IEEE Journal on Selected Areas in Communications, vol. 38, no. 10, pages 2260–2270, 2020. (Cited in pages 75 and 93.)

[Schaller 2017] Sibylle Schaller and Dave Hood. *Software defined networking architecture standardization.* Computer Standards & Interfaces, vol. 54, pages 197–202, 2017. (Cited in page 23.)

[Shah 2001] Devavrat Shah, Sundar Iyer, Balaji Prabhakar and Nick McKeown. *Analysis of a statistics counter architecture.* In HOT 9 Interconnects. Symposium on High Performance Interconnects, pages 107–111. IEEE, 2001. (Cited in page 30.)

[Shahriari 2015] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams and Nando De Freitas. *Taking the human out of the loop: A review of Bayesian optimization.* Proceedings of the IEEE, vol. 104, no. 1, pages 148–175, 2015. (Cited in pages x, 83, 84, 104, and 109.)

[Shone 2018] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai and Qi Shi. *A deep learning approach to network intrusion detection.* IEEE transactions on emerging topics in computational intelligence, vol. 2, no. 1, pages 41–50, 2018. (Cited in page 60.)

[Shpiner 2016] Alexander Shpiner and Eitan Zahavi. *Race cars vs. trailer trucks: Switch buffers sizing vs. latency trade-offs in data center networks.* In 2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI), pages 53–59. IEEE, 2016. (Cited in page 82.)

[Singh 2015] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano*et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network.* ACM SIGCOMM computer communication review, vol. 45, no. 4, pages 183–197, 2015. (Cited in pages 54 and 65.)

[So-In 2009] Chakchai So-In. *A survey of network traffic monitoring and analysis tools.* Cse 576m computer system analysis project, Washington University in St. Louis, 2009. (Cited in page 19.)

[Sony 2000] CSL Sony and Kenjiro Cho. *Traffic data repository at the WIDE project.* In Proceedings of USENIX 2000 Annual Technical Conference: FREENIX Track, pages 263–270, 2000. (Cited in page 38.)

[Stampa 2017] Giorgio Stampa, Marta Arias, David Sánchez-Charles, Victor Muntés-Mulero and Albert Cabellos. *A deep-reinforcement learning approach for software-defined networking routing optimization.* arXiv preprint arXiv:1709.07080, 2017. (Cited in page 59.)

[Su 2014] Zhiyang Su, Ting Wang, Yu Xia and Mounir Hamdi. *FlowCover: Low-cost flow monitoring scheme in software defined networks.* In 2014 IEEE Global Communications Conference, pages 1956–1961. IEEE, 2014. (Cited in page 44.)

[Suárez-Varela 2017] José Suárez-Varela and Pere Barlet-Ros. *Reinventing NetFlow for OpenFlow Software-Defined Networks.* arXiv preprint arXiv:1702.06803, 2017. (Cited in page 45.)

[Sultana 2019] Nasrin Sultana, Naveen Chilamkurti, Wei Peng and Rabei Alhadad. *Survey on SDN based network intrusion detection system using machine learning approaches.* Peer-to-Peer Networking and Applications, vol. 12, no. 2, pages 493–501, 2019. (Cited in page 60.)

[Sun 2016] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu and Bruno Sinopoli. *CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction.* In Proceedings of the 2016 ACM SIGCOMM Conference, pages 272–285, 2016. (Cited in page 58.)

[Taherizadeh 2018] Salman Taherizadeh, Andrew C Jones, Ian Taylor, Zhiming Zhao and Vlado Stankovski. *Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review.* Journal of Systems and Software, vol. 136, pages 19–38, 2018. (Cited in page 25.)

[Tangari 2018] Gioacchino Tangari, Daphne Tuncer, Marinos Charalambides, Yuanshunle Qi and George Pavlou. *Self-adaptive decentralized monitoring in software-defined networks.* IEEE Transactions on Network and Service Management, vol. 15, no. 4, pages 1277–1291, 2018. (Cited in pages 25, 30, 45, and 46.)

[Tayyaba 2020] Sahrish Khan Tayyaba, Hasan Ali Khattak, Ahmad Almogren, Munam Ali Shah, Ikram Ud Din, Ibrahim Alkhalifa and Mohsen Guizani. *5G vehicular network resource management for improving radio access through machine learning.* IEEE Access, vol. 8, pages 6792–6800, 2020. (Cited in page 59.)

[Trihinas 2018] Demetris Trihinas, George Pallis and Marios Dikaiakos. *Low-cost adaptive monitoring techniques for the Internet of Things.* IEEE Transactions on Services Computing, 2018. (Cited in page 45.)

[Tsai 2018] Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu and Chu-Sing Yang. *Network monitoring in software-defined networking: A review.* IEEE Systems Journal, vol. 12, no. 4, pages 3958–3969, 2018. (Cited in pages 19 and 25.)

[Van Adrichem 2014] Niels LM Van Adrichem, Christian Doerr and Fernando A Kuipers. *Opennetmon: Network monitoring in openflow software-defined networks.* In 2014 IEEE Network Operations and Management Symposium (NOMS), pages 1–8. IEEE, 2014. (Cited in page 46.)

[Vasudevan 2009] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G Andersen, Gregory R Ganger, Garth A Gibson and Brian Mueller. *Safe and effective fine-grained TCP retransmissions for datacenter communication.* ACM SIGCOMM computer communication review, vol. 39, no. 4, pages 303–314, 2009. (Cited in page 64.)

[Version] OpenFlow Switch Specification Version. *1.5. 1 (Protocol version 0x06), December, 2014.* (Cited in page 23.)

[Villamizar 1994] Curtis Villamizar and Cheng Song. *High performance TCP in ANSNET.* ACM SIGCOMM Computer Communication Review, vol. 24, no. 5, pages 45–60, 1994. (Cited in page 82.)

[Wang 2016] Pu Wang, Shih-Chun Lin and Min Luo. *A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs.* In 2016 IEEE International conference on services computing (SCC), pages 760–765. IEEE, 2016. (Cited in page 59.)

[Wang 2017] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao and Junchen Jiang. *Machine learning for networking: Workflow, advances and opportunities.* IEEE Network, vol. 32, no. 2, pages 92–99, 2017. (Cited in pages ix, 59, 65, 67, and 104.)

[Wang 2018] Kun Wang, Qihua Zhou, Song Guo and Jiangtao Luo. *Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey.* IEEE Communications Surveys & Tutorials, vol. 20, no. 4, pages 3560–3580, 2018. (Cited in page 55.)

[Wiener 1948] Norbert Wiener. Cybernetics or control and communication in the animal and the machine. Technology Press, 1948. (Cited in page 2.)

[Winstein 2013] Keith Winstein and Hari Balakrishnan. *Tcp ex machina: Computer-generated congestion control.* ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pages 123–134, 2013. (Cited in pages 58 and 79.)

[Xia 2016] Wenfeng Xia, Peng Zhao, Yonggang Wen and Haiyong Xie. *A survey on data center networking (DCN): Infrastructure and operations.* IEEE communications surveys & tutorials, vol. 19, no. 1, pages 640–656, 2016. (Cited in page 53.)

[Xiao 2015] Peng Xiao, Wenyu Qu, Heng Qi, Yujie Xu and Zhiyang Li. *An efficient elephant flow detection with cost-sensitive in SDN.* In 2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom), pages 24–28. IEEE, 2015. (Cited in page 59.)

[Xie 2018] Junfeng Xie, F Richard Yu, Tao Huang, Renchao Xie, Jiang Liu and Yunjie Liu. *A Survey of Machine Learning Techniques Applied to Software*

Defined Networking (SDN): Research Issues and Challenges. IEEE Communications Surveys & Tutorials, 2018. (Cited in pages 24, 59, and 75.)

[Xu 2017] Chenhan Xu, Kun Wang and Mingyi Guo. *Intelligent resource management in blockchain-based cloud datacenters*. IEEE Cloud Computing, vol. 4, no. 6, pages 50–59, 2017. (Cited in page 52.)

[Xu 2019] Yang Xu, Shikhar Shukla, Zehua Guo, Sen Liu, Adrian S-W Tam, Kang Xi and H Jonathan Chao. *RAPID: Avoiding TCP Incast Throughput Collapse in Public Clouds With Intelligent Packet Discarding*. IEEE Journal on Selected Areas in Communications, vol. 37, no. 8, pages 1911–1923, 2019. (Cited in pages 56 and 64.)

[Yu 2013] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang and Harsha V Madhyastha. *Flowsense: Monitoring network utilization with zero measurement cost*. In International Conference on Passive and Active Network Measurement, pages 31–41. Springer, 2013. (Cited in pages 28, 43, and 45.)

[Yu 2019] Minlan Yu. *Network telemetry: towards a top-down approach*. ACM SIGCOMM Computer Communication Review, vol. 49, no. 1, pages 11–17, 2019. (Cited in page 98.)

[Zhang 2011] Jiao Zhang, Fengyuan Ren and Chuang Lin. *Modeling and understanding TCP incast in data center networks*. In 2011 Proceedings IEEE INFOCOM, pages 1377–1385. IEEE, 2011. (Cited in pages 56, 69, and 74.)

[Zhang 2012] Yan Zhang and Nirwan Ansari. *On architecture design, congestion notification, TCP incast and power consumption in data centers*. IEEE Communications Surveys & Tutorials, vol. 15, no. 1, pages 39–64, 2012. (Cited in page 55.)

[Zhang 2013] Ying Zhang. *An adaptive flow counting method for anomaly detection in SDN*. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, pages 25–30. ACM, 2013. (Cited in pages 30 and 46.)

[Zhang 2014] Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou and Jie Wu. *Robust network traffic classification*. IEEE/ACM transactions on networking, vol. 23, no. 4, pages 1257–1270, 2014. (Cited in page 58.)

[Zhang 2016] Zhen Zhang, Yuhui Deng, Geyong Min, Junjie Xie and Shuqiang Huang. *ExCCC-DCN: A highly scalable, cost-effective and energy-efficient data center structure*. IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 4, pages 1046–1060, 2016. (Cited in page 52.)

[Zhang 2018] Jiao Zhang, F Richard Yu, Shuo Wang, Tao Huang, Zengyi Liu and Yunjie Liu. *Load balancing in data center networks: A survey.* IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pages 2324–2352, 2018. (Cited in pages 54 and 60.)

[Zhao 2015] Shuai Zhao, Mayanka Chandrashekar, Yugyung Lee and Deep Medhi. *Real-time network anomaly detection system using machine learning.* In 2015 11th International Conference on the Design of Reliable Communication Networks (DRCN), pages 267–270. IEEE, 2015. (Cited in page 59.)

[Zhao 2019] Yanling Zhao, Ye Li, Xinchang Zhang, Guanggang Geng, Wei Zhang and Yanjie Sun. *A survey of networking applications applying the software defined networking concept based on machine learning.* IEEE Access, vol. 7, pages 95397–95417, 2019. (Cited in page 59.)

[Zhou 2018] Donghao Zhou, Zheng Yan, Yulong Fu and Zhen Yao. *A survey on network data collection.* Journal of Network and Computer Applications, vol. 116, pages 9–23, 2018. (Cited in page 19.)