Intelligent Probing for SDN Monitoring

Edwin Ferney Castillo Quintero

Dissertation of Master in Telematics Engineering

Advisor:
Ph.D. Oscar Mauricio Caicedo Rendon

Universidad del Cauca
Faculty of Electronic and Telecommunications Engineering
Department of Telematics
Line of Research in Advanced Services of Telecommunications
Popayán, May 2020

Edwin Ferney Castillo Quintero

Intelligent Probing for SDN Monitoring

A dissertation submitted to the Faculty of Electronic and Telecommunications Engineering of the Universidad del Cauca for the Degree of:

Master in Telematics Engineering

Advisor:
Oscar Mauricio Caicedo Rendon
Ph.D. in Computer Science

Popayán, May 2020

I've been a fortunate man in life, nothing has come easy.

Sigmund Freud

Acknowledgment

Acknowledgments are written in Spanish because of my family. Ha sido un largo camino hasta llegar a este punto. Quiero aprovechar estas líneas para agradecer a todas las personas que me han acompañado, en los buenos y en los malos momentos, a lo largo de mi etapa universitaria y cuyo desenlace tiene lugar con este trabajo de Maestría

A mis padres, Graciela Quintero y Álvaro Castillo por su apoyo incondicional, por sus consejos, por su cariño, por sus valores y por ser todos ellos grandes ejemplos de perseverancia y animarme a seguir hacia delante en cada etapa de mi vida; su dedicación y sacrificio han sido la base para mi progreso.

A mis hermanos, no solo por estar presentes aportando buenas cosas a mi vida, sino por los grandes momentos de felicidad y de diversas emociones que siempre me han causado. A mi hermano Albert por ser mi inspiración y motivante para mejorar día a día.

A mí querida esposa Esledin por su apoyo incondicional, por sus consejos, por su amor, por siempre estar ahí; sin ti lograrlo hubiese sido más difícil. A mis dos grandes tesoros José y Tefy, el mejor regalo que haya podido recibir de parte de Dios; gracias a ellos por ser la felicidad de mi vida.

A mis amigos, a los de toda la vida y a todos aquellos que he tenido el placer de conocer durante la carrera y la Maestría. Porque sin ustedes estos años no tendrían el significado que han tenido para mí. Gracias por todo el apoyo y por los buenos momentos que hemos compartido y que seguiremos compartiendo.

Y, por supuesto, a mi tutor PhD Oscar Mauricio Caicedo Rendón. Porque sin él no habría sido posible la realización de este trabajo. Me gustaría destacar su apoyo, porque siempre han estado ahí cuando lo he necesitado. Además siempre me ha proporcionado invaluables consejos y se ha preocupado por que se hicieran las cosas de la mejor forma posible. Indudablemente por su influencia hoy soy mejor persona, mejor padre y mejor investigador.

Abstract

Traffic Monitoring assists in achieving the stability of networks by observing and quantifying their behavior. A proper traffic monitoring solution requires the accurate and timely collection of flow statistics. Many approaches have been proposed to monitor Software-Defined Networks. However, these approaches have diverse shortcomings. First, they are unconcerned about the trade-off between the probing interval and the Monitoring Accuracy (MA). Second, they lack intelligent mechanisms intended to optimize this trade-off by learning from network behavior. This master dissertation introduces an approach, called IPro, to address these shortcomings. IPro is formed by an architecture that follows the Knowledge-Defined Networking paradigm, an algorithm based on Reinforcement Learning, and an IPro prototype. In particular, IPro uses Reinforcement Learning to determine the probing interval that keeps within thresholds (target values) the Control Channel Overhead (CCO) and the Extra CPU Usage of the Controller (CUC). An extensive quantitative evaluation corroborates that IPro is an efficient approach for SDN Monitoring regarding CCO, CCU, and MA.

Keywords: Knowledge-Defined Networking, Machine Learning, Probing Interval, Software-Defined Networking, Traffic Monitoring

Resumen

La monitorización de tráfico ayuda a lograr la estabilidad de las redes al observar y cuantificar su comportamiento. Una solución de monitorización de tráfico adecuada requiere la recopilación precisa y oportuna de estadísticas de flujo. Diversos investigadores han propuesto multiples enfoques para monitorear Redes Definidas por Software (Software-Defined Networks - SDN). Sin embargo, estos enfoques tienen algunas deficiencias. En primer lugar, no les preocupa el balance entre el intervalo de sondeo y la precisión de monitoreo (Monitoring Accuracy - MA). En segundo lugar, carecen de mecanismos inteligentes destinados a optimizar este balance al aprender del comportamiento de la red. Esta disertación de maestría introduce un enfoque, llamado IPro, para abordar estas deficiencias. IPro está formado por una arquitectura que sigue el paradigma de las Redes Definidas por el Conocimiento (Knowledge-Defined Networking - KDN), un algoritmo basado en el Aprendizaje por Refuerzo (Reinforcement Learning - RL) y un prototipo de IPro. En particular, IPro utiliza RL para determinar el intervalo de sondeo que mantiene dentro de umbrales (valores objetivo) la sobrecarga del canal de control (Control Channel Overhead - CCO) y el uso adicional de la CPU del controlador (CPU Usage of the Controller - CUC). Una extensa evaluación cuantitativa corrobora que IPro es un enfoque eficiente para el monitoreo de SDN con respecto a CCO, CCU y MA.

Content

List of abbreviations and Acronyms xv

List of Figures xvii

List of Tables xviii

1. Introduction 1
- 1.1. Objectives . . . 2
  - 1.1.1. General Objective . . . 2
  - 1.1.2. Specific Objectives . . . 2
- 1.2. Research Contributions . . . 2
- 1.3. Methodology and Organization . . . 3

2. Background 5
- 2.1. Software-Defined Networking . . . 5
- 2.2. Traffic Engineering . . . 7
- 2.3. Network Monitoring . . . 8
  - 2.3.1. Monitoring Operations . . . 8
  - 2.3.2. Network Monitoring Techniques . . . 9
- 2.4. Machine Learning . . . 11
- 2.5. Final Remarks . . . 13

3. State-of-Art 15
- 3.1. SDN Monitoring . . . 15
- 3.2. Research Gaps . . . 18
- 3.3. Final Remarks . . . 19

4. IPro 21
- 4.1. Motivating Scenario . . . 21
- 4.2. Fundamentals . . . 22
  - 4.2.1. Knowledge-Defined Networking . . . 22
  - 4.2.2. Reinforcement Learning . . . 23
- 4.3. Overview . . . 26

4.4. Architectural Layers and Elements . . . 27
4.4.1. Knowledge Plane . . . 27
4.4.2. Control Plane . . . 29
4.4.3. Management Plane . . . 30
4.4.4. Data Plane . . . 30
4.5. Probing Algorithm . . . 31
4.5.1. Assumptions . . . 31
4.5.1.1. Reward . . . 31
4.5.1.2. Space of Actions . . . 32
4.5.1.3. Space of States . . . 32
4.5.1.4. Statistics Collection . . . 32
4.5.1.5. Control Channel Overhead . . . 33
4.5.1.6. CPU Usage of the Controller . . . 34
4.5.1.7. Monitoring Accuracy . . . 34
4.5.2. Functioning . . . 34
4.5.3. Computational Complexity . . . 36
4.5.4. IPro Interactions . . . 36
4.5.4.1. Statistics Collection Process . . . 36
4.5.4.2. Probing Interval Optimization Process . . . 38
4.6. Final Remarks . . . 39

5. Evaluation **41**
5.1. Setup . . . 41
5.1.1. Test Environment . . . 41
5.1.2. Prototype . . . 42
5.1.3. Space of States . . . 44
5.1.3.1. Control Channel Overhead . . . 44
5.1.3.2. CPU Usage of the Controller . . . 45
5.1.3.3. Monitoring Accuracy . . . 45
5.1.3.4. Spaces Discretization . . . 46
5.2. Intelligent Probing Behavior . . . 47
5.3. Comparison . . . 50
5.3.1. After Converging . . . 50
5.3.2. Before Converging . . . 51
5.3.3. Qualitative Analysis . . . 51
5.4. Final Remarks . . . 53

6. Conclusions **55**
6.1. Answering the Research Question . . . 56
6.2. Contributions . . . 56

6.3. Future work 57

Bibliography **58**

Appendices **69**

A. Appendix A - Scientific Production **69**
A.1. Papers: accepted and on reviewing 69
A.1.1. Accepted 69
A.1.2. On Revision 69

B. Appendix B - Scripts Developed **70**
B.1. Intelligent Probing Repository 70

List of Abbreviations and Acronyms

Abbreviations

Abbreviation	**Term**
AP	Application Plane
AHC	Adaptive Heuristic Critic
COC	Control Channel Overhead
CP	Control Plane
CUC	CPU Usage of the Controller
DP	Data Plane
EWBI	East/Westbound Interface
FM	Flow Management
FT	Fault Tolerance
IETF	Internet Engineering Task Force
InP	Infrastructure Provider
IPro	Intelligent Probing
ISP	Internet Services Provider
KP	Knowledge Plane
KPI	Key Performance Indicator
MA	Monitoring Accuracy
KDN	Knowledge-Defined Networking
MDP	Markov Decision Process
MI	Management Interface
ML	Machine Learning
MP	Management Plane
NBI	NorthBound Interface
NFV	Network Functions Virtualization
NM	Network Management
PFC	Per-Flow Collection
PSC	Per-Switch Collection
PPA	Periodic Probing Approach
RL	Reinforcement Learning

QoS	Quality of Service
SBI	SouthBound Interface
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SL	Supervised Learning
SLA	Service Level Agreement
TAC	Traffic Analysis/Characterization
TE	Traffic Engineering
TU	Topology Update
UL	Unsupervised Learning

Symbols

Symbol	**Term**
ω	Policy 1
χ	Policy 2

List of Figures

1-1. Thesis phases 3

2-1. High-level SDN architecture 5
2-2. Classification of network monitoring operations (adapted from [1]) 8
2-3. Push-based monitoring 9
2-4. Pull-based monitoring 10
2-5. RL Process 13

4-1. High-level KDN architecture (adapted from [2]) 23
4-2. IPro - High-Level Operation 26
4-3. IPro Architecture 27
4-4. The RL-agent General Model 28
4-5. Sequence diagram of statistics collection process 37
4-6. Sequence diagram of probing interval optimization process 38

5-1. Test Environment 42
5-2. IPro - Prototype 43
5-3. CCO Variation 44
5-4. CUC Variation 45
5-5. MA of throughput 46
5-6. TCP errors generated by Monitoring Interference 47
5-7. Behavior of the CCO, CUC, MA, and Probing Interval 48
5-8. Behavior of the CPU usage and memory of RL-agent 49

List of Tables

3-1. Traffic Monitoring in SDN – H → High and L → Low 18

5-1. Comparison after converging 50
5-2. Comparison before converging 51
5-3. Comparison between IPro and other adaptive methods – H → High and L → Low 52

1. Introduction

Software-Defined Networking (SDN) is a paradigm that promotes a flexible architecture for fast and easy configuration of network devices [3]. SDN is characterized by the network programmability, and the centralization of the control functions in the controller. Thanks to these features, the controller can perform fine-grained Network Management (NM) [4]. Nonetheless, such features are not enough to guarantee an appropriate network behavior when traffic reach unexpected levels [5, 6]. In this sense, Traffic Engineering (TE) is an important tool to assist the SDN operation [7]. TE encompasses measuring and managing the network traffic, aiming at improving the utilization of network resources and enhancing the Quality of Service (QoS). TE requires an efficient Traffic Monitoring that allows the accurate and timely collection of flow statistics [8].

SDN-enabled switches can measure different per-flow traffic statistics, such as byte and packet counters, duration per second, hard timeout, and lifetime. There are two ways in which the SDN controller can retrieve traffic statistics from the underlying switches: push-based and pull-based. In the push-based approach [9, 10], the controller passively receives reports from switches. This approach has some drawbacks. First, this method requires additional hardware and software components in the switches. Second, when the traffic varies dynamically, the switches frequently detect no-matching packets in the flow table and, as a result, massive statistical reports are sent to the controller. These massive reports can cause significant Control Channel Overhead (CCO) [11] and an Extra CPU Usage of the Controller (CUC) [12]. Third, the additional hardware and software elements can raise security issues [13]. In the pull-based approach, the controller retrieves flow statistics from the switches using Read-State messages. This approach provides more flexibility that the push-based approach, since it can asynchronously communicate with the switches and request specific information, thus controlling the size of the statistical reports. Besides, this approach does not require changes in the software and hardware of switches. For these reasons, this master dissertation focus on the pull-based approach.

CCO can also appear in the pull-based approach due to the probing interval. This overhead can lead to overload the controller (*i.e.*, CUC) and significantly interfere with essential SDN functions, such as packet forwarding and route updating. Several research approaches have been conducted to deal with CCO and CUC [14, 15, 16, 17, 18, 19, 20, 21, 12, 22, 23, 24, 25,

26]. In particular, [14, 15, 16, 17, 18] reduce CCO by using adaptive techniques, wildcards, threshold-based methods, and routing information at expenses of decreasing the Monitoring Accuracy (MA). Other approaches diminish CCO by adding modules or modifying flow tables in the switches [19, 12, 20, 21] and by adding distributed controllers [22, 23, 24, 25, 26]. Thus, these works reduce CCO but they increase the operational costs. Furthermore, it is noteworthy that, in pull-based solutions the trade-off between probing interval and MA has not been studied enough. Besides, few intelligent mechanisms have been proposed for optimizing such a trade-off by learning from network behavior. Therefore, the goal of this master dissertation is to investigate a practical approach (*i.e.*, in terms of CCO, CUC, and MA) for intelligent probing in SDN. To achieve this goal, this dissertation raises the following research question.

How to intelligently probing SDN with a high accuracy and with a negligible network overhead?

Hypothesis: The Machine Learning allows intelligent probing on SDN, improving the accuracy traffic monitoring and reducing the corresponding overhead.

1.1. Objectives

1.1.1. General Objective

To introduce a mechanism for intelligent probing in SDN by Machine Learning techniques.

1.1.2. Specific Objectives

- To design a mechanism based on Machine Learning for intelligent probing in SDN.
- To implement a prototype of the proposed mechanism.
- To evaluate the mechanism through of a prototype in an SDN emulated scenario according to its network monitoring accuracy and network overhead.

1.2. Research Contributions

The investigation about a practical approach for intelligent probing in SDN led to the following major contributions.

- A KDN-based architecture that provides an efficient solution for tuning the probing interval in SDN. This tuning keeps CCO and CUC within predefined thresholds and provides an acceptable MA.

- A RL-based algorithm that determines the probing interval considering network traffic variations, CCO, and CUC.
- An IPro prototype that implements the proposed architecture.

The above-mentioned contributions were reported to the scientific community through paper submissions to renowned journals (see Appendix A).

- A paper published in the journal Computer Networks. Colciencias index: A1.
- A paper submitted to the journal IEEE Latin America Transactions. Colciencias index: A2.

1.3. Methodology and Organization

Figure **1-1** depicts the phases of the scientific research process followed in this master dissertation: Problem Statement, Hypothesis Construction, Experimentation, Conclusion, and Publication. In Problem Statement, the research question has been identified and defined. In Hypothesis Construction, the hypothesis and associated fundamental question have been formulated. Furthermore, in such a phase, the conceptual and technological proposals have been defined and carried out. In Experimentation, the hypothesis and evaluation results have been tested and analyzed, respectively. In Conclusion, conclusions and future works have been outlined. Note that Hypothesis Construction has been refed after Experimentation and Conclusion. In Publish Findings, papers for renowned conferences and journals have been submitted and published. This document was also written during such last phase.

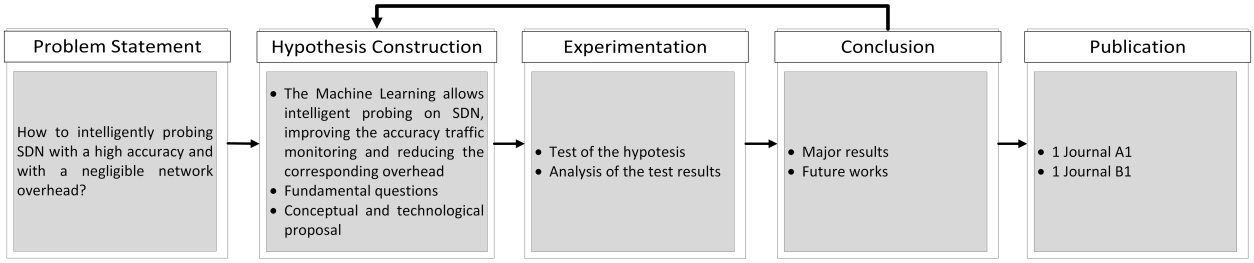

Figure 1-1.: Thesis phases

The organization of this document reflects the phases outlined above.

- **This introductory chapter** presents the problem definition, raises the hypothesis, summarizes the contributions, and describes the overall structure of this master dissertation.
- **Chapter 2** reviews research about SDN, TE, Network Monitoring, ML, and KDN.

- **Chapter 3** presents the related works about SDN monitoring.
- **Chapter 4** presents a motivating scenario for the proposed approach. Also, this chapter introduces IPro, its architectural elements, and algorithmic representation.
- **Chapter 5** describes the experiments conducted to test the hypothesis, discusses the corresponding results, and presents implementation highlights.
- **Chapter 6** presents conclusions about the hypothesis and the fundamental questions, as well as opportunities for future works.
- **Appendix** includes the list of papers in which the major results obtained during the development of this master dissertation have been published or submitted.

2. Background

The goal of this chapter is to present the background of the leading research topics touched in this master dissertation. In this way, this chapter starts showing the SDN concept and its architecture. After, this chapter reviews the TE concept and its use in SDN domains, including the basics of network monitoring. This chapter finishes discussing the ML concept.

2.1. Software-Defined Networking

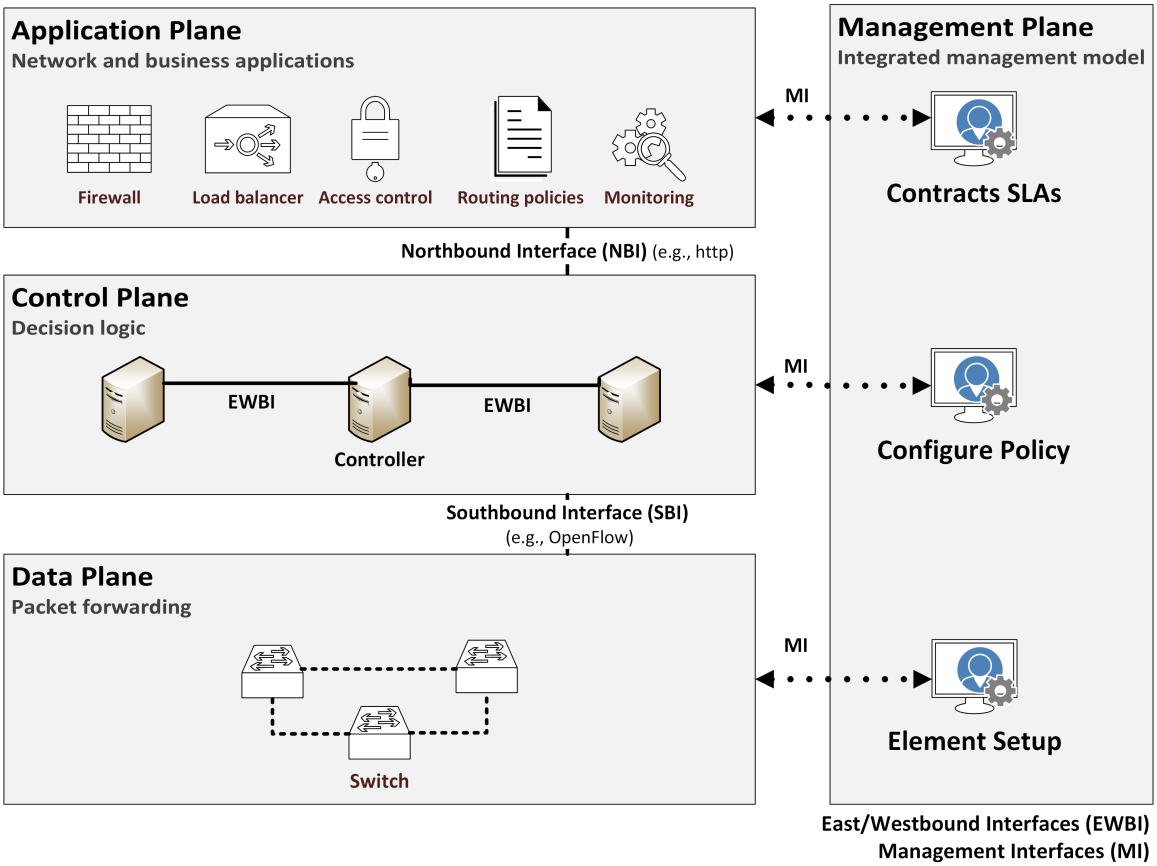

Figure 2-1.: High-level SDN architecture

SDN represents one of the most well-known and attractive trends in academic and industry

for defining the architecture of future networks [6, 27]. SDN has some distinguishing features that define how it is different from traditional networking architecture. These features include [28, 29]: (*i*) clear separation of the control and forward function, (*ii*) centralization of the control function, (*iii*) implementation of the control function in software, (*iv*) open standards, and (*v*) Flow-based. These features make the SDN architecture more flexible, scalable, efficient and adaptable to the changing needs of the business [5]. Furthermore, they make SDN a propitious scenario for efficiently and intelligently implementing monitoring techniques, particularly for TE.

Overall, SDN introduces an architecture with four planes [30]: management, application, control, and data (*cf.* Figure **2-1**). All these planes communicate with each other through interfaces. In particular, the Management Plane (MP) uses a set of Management Interfaces (MI) to exchange information and to control the other planes. The Application Plane (AP) communicates its network requirements to the Control Plane (CP) by NorthBound Interfaces (NBI). The CP defines East/Westbound Interfaces (EWBI) that enable to deploy a distributed controller for coping with large-scale and wide-area networks. The Data Plane (DP) communicates with CP by SouthBound Interfaces (SBI). Ideally, all these interfaces should be standardized to allow easy replacement of devices and technologies. In practice, the OpenFlow protocol is the current *de-facto* standard for the SBI because of its widespread use by vendors and research. All other interfaces are undergoing discussion and development.

- **Management Plane** contains one or more solutions responsible for managing and coordinating each plane individually (*e.g.*, network monitoring, performance management, configuring/planning resources, and enforcing both policies and contracts).

- **Application Plane** contains one or more applications that can serve different purposes (*e.g.*, firewall, load balancer, access control, routing policies, and monitoring). Each application has access to a set of resources of one or more controllers through NBIs.

- **Control Plane** translates the requirements from AP at a specific network policy and enforces it over network elements through SBI. This plane contains one or more controllers (*e.g.*, Floodlight, NOX, POX, and Ryu) that handle and coordinate the network devices.

- **Data Plane** contains a set of programmable network devices responsible for storing, forwarding and processing data packets. This plane depends on CP and MP to populate the forwarding tables and update their configuration.

2.2. Traffic Engineering

TE is emerging as an essential tool for selecting the optimal paths that different flows should follow to optimize resource utilization and satisfy the QoS requirements of each flow [7, 31]. According to the Internet Engineering Task Force (IETF), TE aims to evaluate and optimize network performance, QoS, and user experience of operational IP networks [27, 32, 33]. TE, in SDN, focuses on [7]:

- **Flow Management (FM)** controls network resources appropriately when traffic congestion occurs in network nodes. FM maps and controls the traffic flow to steer traffic most efficiently. For example, when a switch receives flows that do not match any rule in the flows table; it forwards these flows to the controller. The controller analyzes this flow and decides to install it a new forwarding rule in the switches or to remove it. If the traffic consists of a high number of new flows or not match flows, it can generate a significant network overhead and latency at both CP and DP. FM looks for avoiding network overhead and providing a trade-off between load-balancing and latency.

- **Fault Tolerance (FT)** seeks to ensure the immediate, transparent and graceful recovery of the network when a failure occurs in any of its nodes. FT provides mechanisms that enhance network integrity and adopts policies emphasising network survivable. For example, to increase the networking resiliency of SDN, FT could introduce a mechanism for link or node failures. This mechanism could specify alternative ports and paths that enable the switch to change the forwarding path in the policy-based routing without requiring a round trip to the controller.

- **Topology Update (TU)** manages the capacity of the network to carry out planned changes (*i.e.*, it aims to update the policies of the network in real time and ensure their application in each flow). For example, since the centralized controllers manage all switches, these can dynamically configure the global network policy rules. TU should guarantee consistency of the network policies across the switches so that each packet or flow should be handled by either the old policy or the new policy, but not by the two.

- **Traffic Analysis/Characterization (TAC)** deals with the monitoring and verification of compliance with network performance goals to evaluate and debug the effectiveness of the applied TE methods. TAC focuses on mechanisms for monitoring the network, debugging errors, fault detection, data collection, and so on. In particular, TAC is the essential prerequisite for traffic analysis, and it is closely related to discovery the network failures and the prediction of link congestion.

An essential requirement for achieving TE is to provide accurate and reliable network monitoring (*e.g.*, to perform TE [34], it is necessary to correctly detect large flow aggregates in

minutes and pick better routes for these flows). Many network management tasks such as traffic accounting, load balancing, and performance diagnosis all rely on accurate and timely monitoring of a large variety of traffic at different time-scales [3, 23, 34, 35].

2.3. Network Monitoring

2.3.1. Monitoring Operations

Network Monitoring collects measurements of Key Performance Indicators (KPI) and events (*e.g.*, bandwidth utilization and link status) to process them into more meaningful metrics, called Aggregate Metrics (*e.g.*, average delay and network availability) [36]. Generally, network monitoring can be roughly classified into five phases: *Collection, Preprocessing, Transmission, Analysis*, and *Presentation* of the data [1]. Figure **2-2**, depicts the classification of operation phases in network monitoring.

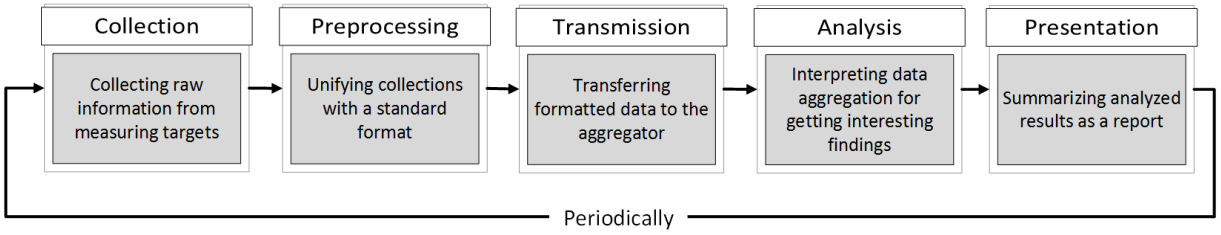

Figure 2-2.: Classification of network monitoring operations (adapted from [1])

- *Collection:* This phase raises three primary considerations namely, means, target, and probing interval. The means refers to how the data are to be collected. The target relates to the devices to be observed, and the probing interval indicates how often the data should be collected from switches.

- *Preprocessing:* This phase is responsible for aggregating and turning the collected data into some statistical format. Furthermore, it helps to itemize and track the measurement results.

- *Transmission:* This phase is responsible for carrying itemized data to the analytic station. The Simple Network Management Protocol (SNMP) and the Network Configuration Protocol (NETCONF) are typical protocols used to exchange messages in the transmission phase. These protocols provide a data delivery interaction between agents and the station.

- *Analysis:* This phase generates statistics and identifies particular events. Some methods perform traffic analysis based on payload or host behavior, whereas other methods

examine communication patterns [37, 38]. The analysis results provide network status information to TE and fault management applications.

- *Presentation:* This phase exports the analysis results in tables, graphs, and reports. For example, Isolani *et al.* [39] proposed an SDN Interactive Manager that provides data visualization via traffic graphs.

A critical task in network monitoring is to achieve a better performance in terms of low overhead (*e.g.*, CCO and CUC) and high MA when measuring the network status. To achieve better performance, it is essential an accurate and timely collection of flow statistics [8].

2.3.2. Network Monitoring Techniques

In SDN, network monitoring techniques can be classified into two categories [36, 40]: push-based (called passive monitoring) and pull-based (called active monitoring). Although the focus of this dissertation is on the pull-based technique, both push-based and pull-based will be discussed briefly.

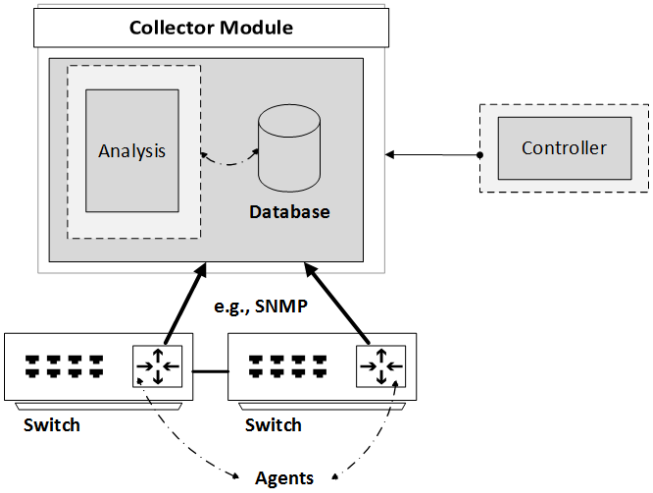

Figure 2-3.: Push-based monitoring

- In the **Push-based** approach [41, 42], the controller observes the statistics information supplied by a collector module without influencing in the performance of the network (*cf.* Figure **2-3**). In this approach, the statistical information is collected by agents placed inside switches (*e.g.*, NetFlow [43], jFlow [44], sFlow [45]), which observe and send the network traffic to the collector module using a monitoring protocol (*e.g.*, SNMP and NETCONF). The collector module is responsible for analyzing and storing this network traffic to make it available to the controller.

Although Push-based approach does not intervene in the performance of the network, several factors hinder its applicability. First, additional elements are necessary in the hardware and software of switches. Second, when the traffic varies dynamically, switches frequently detect no-matching packets in the flow table and, as a result, massive statistical reports are sent to the controller. These massive reports can cause significant CCO [11] and CUC [12]. Third, the push-based approach requires full access to network devices, which could raise privacy and security issues.

- In the **Pull-based** approach, the controller probes one, a set, or all switches using Read-State messages to retrieve statistics from switches (*cf.* Figure **2-4**). There are two variations of Read-State messages: *Request* and *Reply*. The *Request* messages are sent from the controller to switches to request the specific statistical information. The *Reply* messages, on the other hand, are sent from the switches to the controller, delivering the required statistical information.

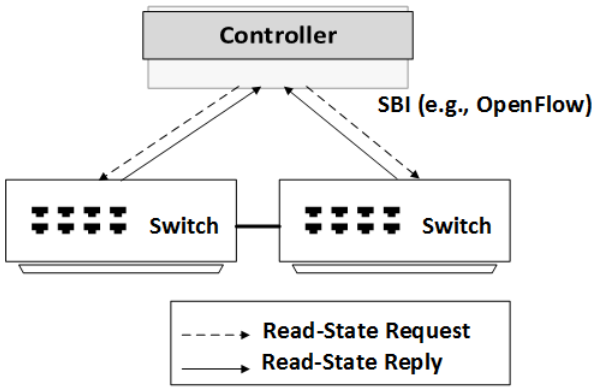

Figure 2-4.: Pull-based monitoring

This approach provides flexibility, since it can asynchronously communicate with switches to request the specific state information and control the size of statistical reports. Nonetheless, CCO can also be introduced in the pull-based approach because of the probing interval. This overhead can lead to overload the controller (*i.e.*, increase CUC) and significantly interfere with essential SDN functions, such as packet forwarding and route updating [46].

This master dissertation focuses on the pull-based approach because:

- It allows requesting specific statistical information from one, a set, or all switches asynchronously, allowing the controller to collect statistical information flexibly and control the size of statistical reports.

- It does not require changes in the software and hardware of switches because the controller only needs Read-State messages, intrinsic in SDN switches, to retrieve statistical information.

2.4. Machine Learning

The quantity of data that flows through communication networks is increasing exponentially. The extraction of knowledge from this data is becoming increasingly important to perform efficient monitoring and management of the network. ML is a tool used to extract useful knowledge from the data and make appropriate decisions [47, 48, 49].

ML includes a set of techniques that can automatically detect patterns in the data (patterns that do not conform to the normal network behavior) to identify previously unseen events and thus to detect network anomalies and to predict future data [50, 51]. The possibilities that emerge from the use of ML in networking context are various [52], some examples are:

- *Network traffic prediction:* Li *et al.* [53] proposed an ML technique that focus is on predicting incoming and outgoing traffic volume on an inter-data center link dominated by elephant flows. Poupart *et al.* [54] explored the use of ML for flow size prediction and elephant flow detection. Chen *et al.* [55] investigated the possibility of reducing the cost of monitoring and collecting traffic volume, by inferring future traffic volume based on flow count only.

- *Resource management:* Bojovic *et al.* [56] designed an ML-based radio admission control mechanism to guarantee QoS for various services, such as voice, data, video and FTP while maximizing radio resource utilization in long term evolution (LTE) networks. Vassis *et al.* [57] proposed an adaptive and distributed admission control mechanism for variable bitrate video sessions, over ad hoc networks with heterogeneous video and HTTP traffic. Quer *et al.* [58] developed an admission control mechanism for VoIP calls in a WLAN. They employed ML to predict the voice call quality as a function of link-layer conditions in the network.

- *Fault management:* Snow *et al.* [59] used ML to estimate the dependability of a 2G wireless network, which is used to characterize availability, reliability, maintainability, and survivability of the network. Lu *et al.* [60] use a manifold learning technique to automatically extract failure features and generate failure prediction. Pellegrini *et al.* [61] proposed an ML-based framework to predict the remaining time to failure of applications.

- *Congestion control:* Liu *et al.* [62] proposed an approach using ML for inferring the cause of packet loss in hybrid wired-wireless networks. Fonseca and Crovella [63] fo-

cused on detecting the presence of packet loss by differentiating Duplicated ACKs caused by congestion losses and reordering events. Jayaraj *et al.* [64] tackled the classification of congestion losses and contention losses in optical burst switching networks.

Furthermore, ML has been used successfully in other domains, including agriculture [65, 66, 67], economics [68, 69], and cybersecurity [70, 71].

ML can be divided into three categories, based on how the learning is achieved [72, 51]:

- **Supervised Learning (SL)** requires a set of instances, commonly known as training data, which are used to define the behavior of algorithm. The training data consists of a set of attributes and an objective variable (also called class), which is intended to be classified or predicted. When the domain of the attribute is categorical[1], the problem is known as classification or pattern recognition, and when the domain of the attribute is numeric[2] value, the problem is known as regression. The most representative algorithms of SL are: Bayesian Networks, Support Vector Machines, k-Nearest Neighbors, Decision Trees, and Neural Networks [48, 52].

- **Unsupervised Learning (UL)** explores the structure of a non-tagged dataset (without target variable). This method reveals unexpected characteristics (patterns) and generates new groups (each with different identifiable properties). The unsupervised learning algorithms are divided into three relevant categories:

  - *Hierarchical Algorithm* aims to obtain a hierarchy of clusters, called dendrogram, that shows how the clusters are related to each other [73].
  - *Density Algorithm* finds the areas with the highest data density, leaving aside the sparsely populated regions, which are considered border points or noise [74].
  - *Clustering Partitional Algorithms* aims to directly obtain a single partition of the collection of items into clusters [73].

- **Reinforcement Learning (RL)** is an iterative process in which an agent learns a decision-making process by interacting with the environment (*cf.* Figure **2-5**). The agent is aware of the state of the environment and takes actions that produce changes of state. For each action, the agent receives a reward depending on how good was the action taken. The goal of the agent is to maximize the total reward it receives over time.

  RL is best suited for making cognitive choices, such as decision making, planning, and scheduling. For instance, RL has been successfully applied to complex problems such

[1] A domain is defined as categorical if it is finite (discrete numerical values) and unordered
[2] A numeric domain consists of real numbers

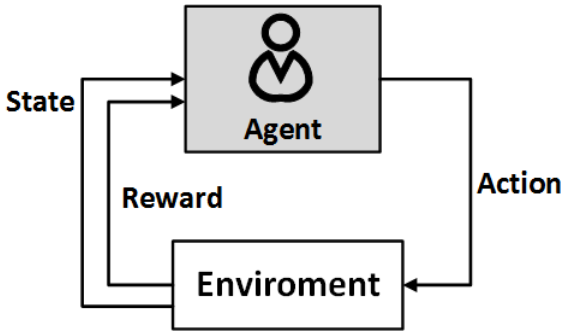

Figure 2-5.: RL Process

as board games [75], job-shop scheduling [76], elevator dispatching [77], and motor control tasks [78], either simulated or real [79].

2.5. Final Remarks

Initially, in this chapter, the concept of the SDN and the architectural features that make the SDN more flexible, scalable, efficient, and adaptable to the changing needs of the business were detailed. Subsequently, the fundamental concepts about TE and network monitoring in the SDN context were described, along with the monitoring techniques related to traffic collection. Finally, the ML concept was introduced, along with its categories and its use in the networking context. This master dissertation considers these concepts for proposing an approach (*cf.* Chapter 4) that focuses on optimizing the probing interval regarding CCO and CUC.

3. State-of-Art

This chapter provides an overview of the paramount efforts around SDN traffic monitoring. In this way, this chapter starts presenting the traffic monitoring proposals for SDN with a higher impact on the research community. This chapter finishes discussing these research works.

3.1. SDN Monitoring

In general, network monitoring is a fundamental topic and an essential requirement for achieving TE [27, 52, 80]. IP networks are usually monitored with mechanisms such as NetFlow [43], jFlow [44], and sFlow [45]. These mechanisms use probes connected to special modules placed inside switches. These probes collect either complete or sampled traffic statistics and send them to a central collector. NetFlow and JFlow are both proprietary solutions and incur a considerable up-front licensing and setup cost to be deployed in a network. sFlow is less expensive to deploy, but it is not widely adopted by the vendors [14]. It is important to highlight that the network monitoring in traditional IP networks can lead to high overhead and significant switch resource consumption [81]. Recent research efforts have harnessed the power of SDN to propose solutions for such monitoring. This section presents some approaches to network monitoring found in the literature.

Raumer *et al.* introduced an application for the Quality of Service (QoS) monitoring, called MonSamp [15]. MonSamp completely moves monitoring capabilities out of the controller and provides fully processed information to the applications by a Northbound API. To traffic collect, MonSamp provides sampling algorithms that can adapt to both current network load and the QoS requirements. In particular, MonSamp suggests decreasing the sampling rate under high traffic load. MonSamp uses thresholds to adjust the amount of monitoring overhead that is forwarded by the switches to allow a certain MA.

Van Adrichem *et al.* proposed OpenNetMon [16] to monitor network throughput, packet loss rates, and network delays continuously. OpenNetMon uses an adaptive probing mechanism to extract statistical information from the switches. The adaptive probing mechanism increases the rate of the queries when the flow rates differ between samples (increases the

MA, CCO, and CUC) and decreases when flows stabilize (reduces the MA, CCO, and CUC).

Tahaei *et al.* proposed a multi-objective network measurement mechanism [17] to overcome the CCO, CUC, and MA in a real-time environment. To strike the trade-off between the MA and CCO, this mechanism uses an elastic probing for the collection of statistics from the switches. The elastic probing adaptively adjusts probing frequency based on the behavior of link utilization. In particular, it increases the probing frequency according to the bandwidth fluctuation of the flows (*i.e.*, to higher bandwidth fluctuation higher probing frequency). Tahaei *et al.* also made an extension to his previous work. This extension proposed a generic architecture for flow measurement in a data-center network, which applies in both single and multiple-controller [26]. The three main features of this architecture are: first, it utilizes local controllers to pull flow statistic and forwards statistics to an upper layer application. Second, it has a coordinator level on top of all the local controllers connecting to the switches. Third, it is implemented as a standard northbound interface, which can utilize both fixed and adaptive polling systems.

Tootoonchian *et al.* proposed OpenTM [18] to estimate the traffic matrix using OpenFlow statistics (*e.g.*, bytes and packet counters). OpenTM proposes several heuristics (*e.g.*, last switch, round robin, uniform random selection, and non-uniform random selection) to (i) choose an optimal set of switches to be monitored for each flow and (ii) keep the trade-off between MA and CCO. After a switch has been selected, OpenTM probes the switch selected continuously for collecting flow level statistics. The choice of a heuristic defines the level of MA and CCO. For instance, the use of the last switch heuristic results in the most accurate traffic matrix but imposes a substantial overhead on edge switches. The price to use the uniform random selection heuristic is to lose some accuracy.

Chowdhury *et al.* proposed a flow measurement framework called PayLess [14]. It is designed as a component of the OpenFlow controller, and it provides a RESTful API for flow statistics collection at different aggregation levels (*e.g.*, flow, packet, and port). In particular, this framework is responsible for parsing request commands from the application level of tasks and transforming these commands into path planning on some switches. PayLess adjusts the probing frequency to balance the CCO and MA. To achieve this balance, Payless relies on OpenTM [18] to select only important switches to be monitored. Nonetheless, instead of continuously probing a switch, PayLess offers an adaptive scheduling algorithm for probing that achieves the same level of accuracy as continuous probing with much less overhead.

Peng *et al.* proposed a traffic management solution (HONE) based on joint statistical information from the OpenFlow network and end hosts [19]. HONE uses software agents placed inside end hosts and a module that interacts with OpenFlow switches. HONE integrates

information from the OpenFlow network and end hosts to present a diverse collection of fine-grained monitoring statistics. Furthermore, HONE offers two techniques to process flow statistics: The first technique, known as the lazy materialization of the measurement data, it uses database-like tables to represent the statistical information collected from hosts and network devices. Lazy materialization allows that both the controller and host agents use the statistical information collected in multiple management tasks. The second technique offers data parallel streaming operators for programming the data-analysis logic. The operators can also be used in a hierarchical fashion for aggregate analysis among multiple hosts.

Phan *et al.* proposed a scalable framework called SDN-Mon [20]. SDN-Mon decouples monitoring from existing forwarding tables to allow more fine-grained and flexible monitoring. This framework uses a controller-side module and a switch-side module. The controller-side module defines a set of monitoring match fields based on the requirements of applications to allow higher flexibility. The switch-side module process the monitoring functionality of the framework to reduce the CCO. Phan *et al.* [25] also proposed a mechanism that supports distributed monitoring capability of SDN-Mon. This extension introduces three additional modules: the switch module, the controller module, and the external module, which allow SDN-Mon can automatically assign the monitoring load to multiple monitoring switches in a balanced way and eliminate duplicated monitoring entries.

Liao *et al.* proposed a solution for latency monitoring called LLDP-looping [21]. LLDP-looping monitors the latency of all network links by using agents placed inside switches and a controller module that interacts with them. The controller module uses a greedy algorithm to inject probe packets (i.e., time-stamped Link Layer Discovery Protocol (LLDP) packets) to a selected set of switches to minimize the CCO. The agents, first, forces to each probe packet to loop around a link for three times, then, calculates the RTT of the LLDP packet.

Jose *et al.* [22] presented a monitoring framework that use secondary controllers to identify and monitor aggregates flows using a small set of rules that changes dynamically with traffic load. In this framework, on the one hand, the switches match packets against a small collection of wildcard rules available in Ternary Content Addressable Memory (TCAM), then the counter of the matched highest priority rule is updated. On the other hand, the controllers only read and analyze the relevant counters from the TCAM with fixed time intervals. Furthermore, the authors proposed a heavy hitters algorithm that identifies large traffic aggregates, striking a trade-off between MA and switch overhead.

Tangari *et al.* proposed a decentralized approach for resources monitoring [23]. This approach achieves high reconfiguration reactivity with acceptable accuracy and negligible CCO. It uses local managers, distributed over the network, to adaptively reconfigure the network

resources (under their scope of responsibility). Furthermore, it uses entities installed on local managers to support a wide range of measurement tasks and requirements regarding monitoring rates and information granularity levels. Tangari *et al.* also extended the previous work by proposing a self-adaptive and decentralized framework for resource monitoring [24]. This framework uses a self-tuning, adaptive monitoring mechanism that automatically adjusts its settings based on the traffic dynamics, which improves the MA.

3.2. Research Gaps

Table 3-1.: Traffic Monitoring in SDN – H → High and L → Low

Work	Accuracy	Overhead	Resources-Cost	Flexibility-Scalability	Description
[12]	H	L	H	L	A heuristic algorithm (Greedy) is used to minimize the CCO, obtain the polling scheme efficiently, and handle flow changes
[14]	H	H	H	H	Adaptive sampling algorithms are used to tune the load level generated by monitoring process
[15]	H	H	H	H	Thresholds are used to adjust the load level generated by monitoring process
[16]	H	H	H	L	An adaptive fetching mechanism monitors per-flow metrics, such as throughput, delay, and packet loss
[17]	H	H	L	H	An adaptive flows statistical collection method is used to adjust the polling intervals
[18]	H	H	L	H	Routing information from the controller and flow forwarding path information are used to monitor the link utilization
[19]	H	L	H	L	Software agents residing on hosts interact with network devices to perform monitoring tasks
[22]	H	L	H	L	A small set of matching rules and secondary controllers are used to identify and monitor aggregate flows
[20]	H	L	H	L	Monitoring is decoupled from existing forwarding tables and uses customized software agents in the switches to process their monitoring functionality
[21]	H	L	H	L	It injects time-stamped LLDP packets into switches to monitor the network latency
[23]	H	L	H	L	Local managers and entities are used to reconfigure the network resources and support monitoring tasks at different granularity level
[24]	H	L	H	L	A self-tuning monitoring mechanism is used to automatically adapt its settings based on the traffic dynamism
[25]	H	L	H	L	Extra modules are included in the switches to distribute the monitoring tasks in a balanced way
[26]	H	L	H	L	A two layers hierarchy of controllers is described. The lowest layer polls the flow statistic and forwards statistics to the top layer. The highest layer coordinates the controllers located at the lowest level

Table **3-1** summarizes relevant works in the field of SDN monitoring, and reveals several facts:

- Several works, such as [12, 19, 20, 21, 22, 23, 24, 25, 26] minimize CCO and improve MA by adding modules, modifying flow tables in the data plane or distributing controllers. As a result, in these works, MA is increased at the expense of an increase in network resources and costs. Furthermore, these works do not support fine-grained monitoring and lack of flexibility and scalability needed to cope with a large amount of flows.

- Other works, such as [14, 15, 16, 17, 18] use adaptive techniques, wildcards, threshold-based methods, and routing information to increase MA. Nevertheless, in these approaches, a network overhead (*i.e.*, imbalance in the Accuracy/Overhead) is generated. Also, the controller is overloaded while collecting the flow information from switches.

Despite the progress in SDN monitoring, existing approaches still have some shortcomings:

- They introduce overhead that degrades the network performance or require substantial economic investments.

- They do not optimize the probing interval by intelligent mechanisms intended to keep CCO and CUC within defined thresholds without compromising MA.

This master dissertation addresses these shortcomings by following the KDN paradigm. In particular, RL is used in the KP of KDN.

So far, in the literature, it has not been proposed approaches-based on RL for SDN monitoring. However, the literature some works that use RL for other network tasks are exposed. Zhang *et al.* proposed Q-placement [82], an RL-based algorithm to optimally decide where to place the network services iteratively. Sampaio *et al.* proposed a model of RL integrated into the Network Functions Virtualization (NFV) architecture using an agent that resides in the orchestrator, which guarantees the flexibility necessary to react to network conditions on demand [83]. Yu *et al.* proposed an RL-based mechanism to achieve universal and customizable routing optimization [84]. Jiang *et al.* used RL to provide an end-to-end adaptive HTTP streaming media intelligent transmission architecture [85]. Wang *et al.* presented a software-defined cognitive network for IoV (Internet of Vehicles) called SDCoR. SDCoR uses RL and SDN to provide an optimal routing policy adaptively through sensing and learning from the IoV environment [86]. Arteaga *et al.* proposed an NFV scaling mechanism based on Q-Learning and Gaussian Processes, which uses an agent to carry out an improvement strategy of a scaling policy to make better decisions based on performance variations [87].

3.3. Final Remarks

In this chapter, several research works that aim to minimize CCO and improve MA were presented. The research works analysis revealed several facts. First, they introduce overhead

that degrades the network performance or require substantial economic investments. Second, they do not optimize the probing interval by intelligent mechanisms intended to keep CCO and CUC within defined thresholds without compromising MA. Third, they do not consider the use of RL for SDN monitoring.

Unlike the works presented in this chapter, this master dissertation considers concepts from the KDN discipline for proposing an approach (*cf.* Chapter 4) that focuses on optimizing the probing interval regarding CCO and CUC.

4. IPro

This section presents a motivating scenario for the approach proposed. Also, this section introduces IPro, its architectural elements, and algorithmic representation.

4.1. Motivating Scenario

Let us assume that an Infrastructure Provider (InP) uses an SDN to offer services to one or more Internet Services Providers (ISPs). In particular, this provider provides its services by creating particularized slices for each ISP. Each slice must meet specific Service Level Agreements (SLAs) between InP and ISP. If a performance degradation occurs in one slice, the services provided by any ISP may also be affected, which can lead to non-compliance with the corresponding SLA. This non-compliance can lead to monetary and legal sanctions for InP.

Considering the above-described scenario, it is necessary an efficient and reliable traffic monitoring approach that accurately and timely collects the statistics of flows; these statistics are indispensable to make decisions timely. There are three options to achieve this monitoring:

- To use specialized software modules (*i.e.*, agents that collect specific traffic) installed into the network devices or distributed controllers. Notwithstanding, this option does not support fine-grained monitoring and lacks flexibility and scalability; especially, in networks with a large number of flows [22].

- To use control messages between switches and the centralized controller when a new flow comes in or upon the expiration of a flow entry in the table flow. This option is inaccurate under dynamic traffic conditions [88].

- To use adaptive probing methods, but up to now, they do not offer a trade-off between MA, CCO, and CUC when the network workload is high. IPro provides an adaptive probing that offers such a trade-off by applying RL.

4.2. Fundamentals

4.2.1. Knowledge-Defined Networking

In 2003, D. Clark *et al.* proposed a new construct, the Knowledge Plane (KP) [89], a distributed cognitive system additional to the traditional planes (data and control) of computer networks that permeates the network. This plane was proposed as a pervasive system that builds and maintains high-level models of what the network is supposed to do. KP was envisioned to rely on ML, aiming to bring many advantages to networking, such as automation (recognize-act) and recommendation (recognize-explain-suggest). Although, KP has the potential to represent a paradigm shift in the way the computer networks are operated, optimized, and troubleshot, up to now, such a plane has not been widely deployed. This constrained deployment is because of diverse shortcomings:

- In traditional networks the learning is partial since the switches and routers only have a partial view and control, avoiding achieve a handle beyond local domain.
- Fifteen years ago, network devices located at DP had limited capabilities of storage and computing.

Nowadays, the shortcomings aforementioned may be overcome because, first, SDN provides a full control and rich view of the network from a logically centralized point. Second, the capabilities of network devices have significantly improved, facilitating the gather of information in real-time about packets and flow-granularity. Therefore, KDN has been proposed as a cooperative paradigm that applies ML to SDN [2], aiming at learning the behavior of the network and, in some cases, automatically operate the network accordingly the learned. Furthermore, they corroborate the feasibility of using KDN for Routing in an Overlay Network and Resource Management in a Network Function Virtualization (NFV) scenario. To sum up, the final goal of KDN is to achieve self-driving networks. Figure **4-1** depicts an overview of the KDN paradigm and its functional planes.

- The DP is composed of network devices of programmable forwarding and is responsible for storing, forwarding and processing data packets. This plane depends on the CP and MP planes to populate the forwarding tables and update their configuration.
- The CP translates the requirements from the KP and AP in a specific network policy. Subsequently, CP is based on this policy to update and program matching and processing rules from the DP forwarding elements by SBI.
- The MP, as well as CP, ensures the correct operation and performance of the network in the long term. It defines the network topology and handles the provision and

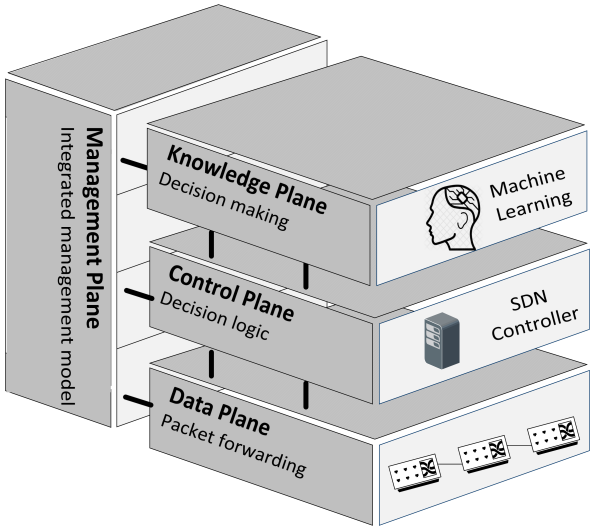

Figure 4-1.: High-level KDN architecture (adapted from [2])

configuration of network devices. MP generates Metadata with information about the network state, events, statistical metrics per flow and per switch (*e.g.*, packet loss, link failure, memory usage, and CPU utilization). The Metadata is sent to CP and KP. CP handles events that require immediate action (*e.g.*, link failure, black-hole or loop detection). KP handles events that require knowledge (*e.g.*, resource planning, optimization, performance management, and verification).

- The KP leverages MP and CP to obtain a rich view and control over the network. It is responsible for learning the behavior of the network and, in some cases, automatically operate the network accordingly. Fundamentally, the KP processes the Metadata generated by the MP, transforms them into knowledge via ML, and uses that knowledge to make decisions (either automatically or through human intervention). It is important to mention that, KP is separated from CP because ML algorithm is generally compute-intensive and it may affect the performance of the control plane.

This master dissertation supposes that RL is a technique useful to maintain MA and decrease the overhead of monitoring in SDN because it allows optimizing the probing interval by interacting with the network itself (*i.e.,* environment in RL terms).

4.2.2. Reinforcement Learning

In RL, an agent learns a decision-making process by interacting with an environment [90]. Formally, in RL, the environment is typically modeled as a finite Markov Decision Processes (MDP) [91] where the agent sends actions and receives outputs (observations and rewards).

In a finite MDP, the agent and environment interact at discrete time steps $t = 0, 1, 2, ..., N$. At each time-step t, the agent receives some representation of the state of the environment, $S_t \in S$, where S is the set of possible states. Based on S_t, the agent selects an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of available actions in the state S_t. The execution of action A_t puts the agent into the new state S_{t+1}. Furthermore, the agent receives a numerical reward from the environment, $R_{t+1} \in R$ at step $t \in N$. Then, the total reward that the agent receives over its lifetime for this particular behavior is:

$$U(s) = \sum_{t=0}^{\infty} \gamma^t R_t \tag{4-1}$$

where $\gamma \in (0, 1]$ is called the discount factor. If $\gamma < 1$, the discounting is used. Otherwise, it is not used.

RL algorithms find an optimal policy $\Pi^* : S \rightarrow A$ that maximizes the expected cumulative reward for every state, using the exploration methods (*e.g.*, ε-greedy, Boltzmann [90] [92]). The main RL features are its capacity to run without any prior knowledge of the environment (here, the monitored network) and make its own decisions in execution time (on-line). Nonetheless, RL requires a training period to capture the environment model before converging to the optimal policy.

Q-learning is one of the most important RL techniques [93] because:

- It was the pioneering RL method used for control purposes.
- It has a learning curve that tends to converge quickly.
- It is the simplest technique that directly calculates the optimal action policy without an intermediate cost evaluation step and without the use of a model *(i.e.*, model-free).
- It has an off-policy learning capability; this means, the agent can learn an optimal policy (called Q-function), even if it does not always choose optimal actions. The only condition is that the agent regularly visits and updates all the (S_t, A_t) pairs.

It is important to highlight that there are other RL strategies, such as Adaptive Heuristic Critic (AHC), Model-free Learning With Average Reward, and some Q-learning variations [94] [95]. These strategies are out of the scope of this master dissertation.

Q-learning [96] [97] relies on an optimal action-value function $Q_t(S_t, A_t)$, called Q-function. In this function, the value is the estimated reward of executing an action A_t in the state

S_t, assuming that the agent will follow the policy that provides the maximum reward. Q-learning starts with an arbitrary Q-function Q_0. At any state S_t, an agent selects an action A_t that determines the transition to the next state S_{t+1} and with the value associated to the pair (S_t, A_t) adjusts the values of Q-function according to:

$$Q_{t+1}(S_t, A_t) \leftarrow (1-\alpha) \cdot Q_t(S_t, A_t) + \alpha \cdot \left[R_{t+1} + \gamma \cdot \max_{\text{A}} Q_t(S_{t+1}, A)\right] \tag{4-2}$$

where R_{t+1} denotes the reward received at time $t+1$, $\alpha \in [0, 1]$ is the learning factor (a small positive number) that determines the importance of the acquired reward. A factor $\alpha = 0$ makes the agent does not learn from the latest (S_t, A_t) pair. In turn, a factor $\alpha = 1$ makes the agent considers the immediate rewards without taking into account the future rewards. $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards. A factor $\gamma = 0$ prohibits the agent from acquiring future rewards. A factor $\gamma = 1$ forces the agent only to consider future rewards. The part between square brackets is the updated value that represents the difference between the current estimate of the optimal Q-value $Q_t(S_t, A_t)$ for a state-action pair (S_t, A_t), and the new estimate $\left[R_{t+1} + \gamma \max_{\text{A}} Q_t(S_{t+1}, A)\right]$.

The Q-function approximates the optimal state-action value function Q^* regardless of the followed policy. It is noteworthy that the updated Q-function Q_t only depends on the previous function Q_{t-1} combined with the experience (S_t, A_t, R_t, S_{t+1}). Thus, Q-learning is both computationally and memory efficient. Nonetheless, if the number of states is high, Q-learning may take much time and require more data to converge (*i.e.,* find the best action for each state). Therefore, in Q-learning is critical to have a concise representation of the environment (*i.e.,* the network model).

To find the Q-function, Q-learning requires an exploration method. The exploration method selects an action to perform at each step, which represents the Q-function. ε-greedy exploration is one of the most used exploration methods [92] [98]. It uses $\varepsilon \in [0, 1]$ as the parameter of exploration to decide which action to perform using $Q_t(S_t, A)$. With this parameter the action is as follows:

$$\mathcal{A} = \begin{cases} \max_{\text{A}} Q_t(S_t, A) & with\, probability\, 1-\varepsilon \\ random\, action & with\, probability\, \varepsilon \end{cases} \tag{4-3}$$

ε-greedy exploration method adds some randomness when deciding between actions. Instead of always selecting the best available action, this method randomly explores other actions with a probability $= \varepsilon$ or chooses the best action (highest Q-value) with a probability $=$

$1 - \varepsilon$. A high value for ε adds randomness to the exploration method, which will make the agent explores other actions more frequently. Randomness is necessary for an agent to learn the optimal policy.

4.3. Overview

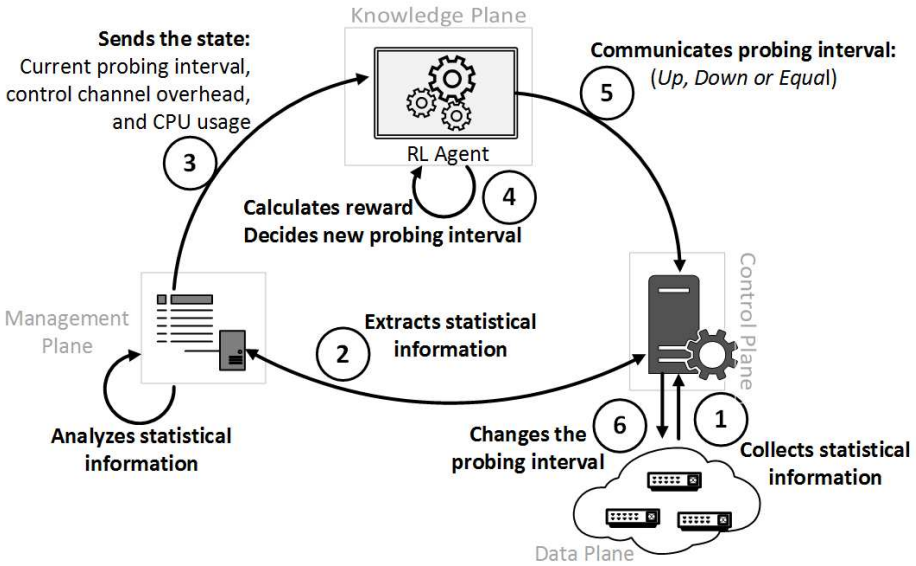

Figure 4-2.: IPro - High-Level Operation

IPro applies RL to optimize the probing interval regarding to CCO and CUC. Figure **4-2** depicts how IPro operates in a high-abstraction level:

(1) The Control Plane (CP) collects statistical information from the Data Plane (DP) at some probing interval. Since this collection of information affects the network behavior, the network falls in a new state.

(2) The Management Plane (MP) extracts these statistics to determine such a new state by analyzing CCO and CUC.

(3) MP sends this new state to the Knowledge Plane (KP).

(4) The RL-agent takes such a state to calculate the reward. It is important to highlight that a low reward indicates high CCO and high CUC. Based on the reward, the RL-agent decides a new probing interval intended to minimize CCO and CUC.

(5) The RL-agent communicates this new probing interval to CP.

⑥ The CP applies this interval that affects the network behavior again. This operation continues until the network administrator decides to stop IPro.

4.4. Architectural Layers and Elements

Figure **4-3** introduces and details the IPro architecture. IPro has four main planes that follow the KDN fundamentals. Next, these plans are detailed.

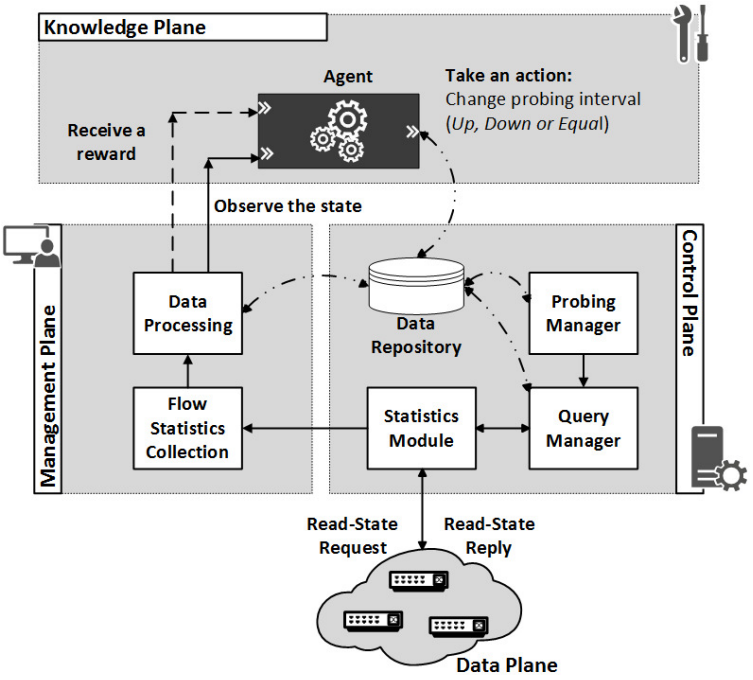

Figure 4-3.: IPro Architecture

4.4.1. Knowledge Plane

This plane obtains a rich view and global control over the network from MP and CP. Overall, KP operates in three steps:

- It organizes the metadata generated by MP.
- It converts that metadata into knowledge by using ML techniques.
- It uses that knowledge to make decisions (either automatically or through human intervention) related to routing, traffic classification, anomalies detection, and so on.

It is essential to highlight that KP is separated from CP because ML algorithm is generally compute-intensive, and it may affect the performance of CP [2]. In this master dissertation, KP is responsible for learning the network behavior and automatically deciding a new probing interval. KP obtains the current network status and controls the probing interval by interacting with MP and CP, respectively. The KP heart is the RL-agent. This agent is in charge of determining the most-rewarding probing strategy intended to maintain CCO and CUC within target values. IPro considers two thresholds ω and χ that are configurable according to network requirements. ω represents the policy 1 and aims at preventing a high CCO. In turn, χ represents the policy 2 and aims at preventing a high CUC. To sum up, the RL-agent handles the monitoring policies by controlling ω and χ.

Control policies are defined to prevent monitoring messages from affecting the performance of the controller and the bandwidth of the control channel. For instance, when CUC exceeds 80% of the CPU capacity, the controller increases its response time, which generates delays and loss [99]. In the same sense, when the use of the control channel bandwidth exceeds 80%, the monitoring messages can interfere with the SDN functions [100].

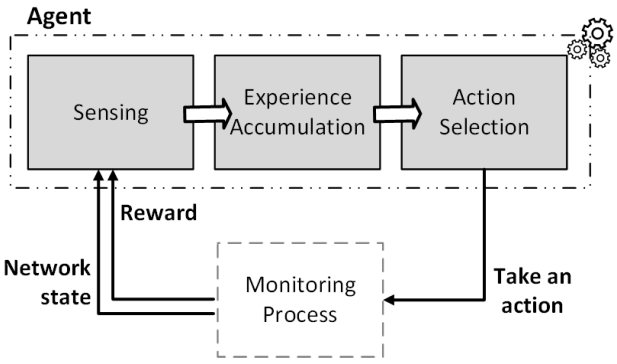

Figure 4-4.: The RL-agent General Model

Figure **4-4** depicts the general model of the RL-agent. This model includes three elements, namely, *Sensing*, *Experience Accumulation*, and *Action Selection*. *Sensing* enables the RL-agent to get the network state and the reward of the IPro monitoring process. The *Experience Accumulation* element guides the policy (i.e., what action to take) of the agent in two steps. In the first one, this element combines the observations made by *Sensing* and defines the information matrix that represents the internal states of the RL-agent. In the second step, *Experience Accumulation* maps these states and assigns rewards to indicate how good or bad these states are according to the accumulated experience. *Action Selection* guides the action policy considering the most-rewarding probing strategy based on the current network state. Section 4.5 depicts the detailed procedure carried out by the RL-agent.

4.4.2. Control Plane

This plane translates the requirements from KP and the Application Plane (AP) in specific network policies. CP uses such policies to update and program matching and processing rules in DP forwarding elements via a SBI, such as OpenFlow [101], Forwarding and Control Element Separation (ForCES) [102], and Protocol-Oblivious Forwarding (POF) [103].

In this master dissertation, CP handles network monitoring by performing the steps described in Algorithm 1.

- CP receives decisions about the new probing interval (I) from KP.
- CP applies these decisions to handle the data collection in the monitored network.
- CP sends Read-State request messages to each switch connected (line 3) to this plane every I seconds (line 5).
- CP receives the Read-State reply messages asynchronously (line 6), which contain the statistical information from the switches.
- CP stores the statistical information (line 7) to maintain a trace of network changes. These steps are repeated indefinitely up to reach a stop criterion (*e.g.,* errors and administrator).

Algorithm 1: Data Collection

```
   Require:
            Probing interval I
1  while not reached stopping criterion do
2      foreach switch ∈ switches do
3          Send a Read-State Request message to switch
4      end
5      Wait for I seconds
6      Receive the Read-State Reply Messages from the switches // These messages contain the
        statistical information;
7      Store the statistical information
8  end
```

CP includes four elements namely, *Probing Manager, Query Manager, Statistics Module, and Data Repository.*

- *Probing Manager* sets up the probing interval according to the decision made by the RL-agent.

- *Query Manager* handles the data collection based on the computed probing interval and the desired aggregation levels (*e.g.*, byte, packet, flow). After this data collection, *Query Manager* merges and stores the statistical information into the data repository. Thus, this information can be used ulteriorly by upper-layer applications.
- *Statistics Module* is a service running on the top of the SDN controller, which is useful to develop customized network measurement applications.
- *Data Repository* stores statistical information of each monitoring operation and maintains a trace of network changes.

It is important to highlight that CP interacts with KP by a Data Repository.

4.4.3. Management Plane

This plane ensures that the network as a whole is running optimally carrying out the Operation, Administration, and Maintenance (OAM) functions [104, 105]. MP defines the network topology and handles the provision and configuration of network devices. Furthermore, it generates metadata with information about the network state, events, statistical metrics per-flow, and per-switch (*e.g.*, packet loss, link failure, memory usage, and CPU utilization).

In this master dissertation, this plane is responsible for extracting the statistical information from CP to provide an analysis of the network state to KP regarding CCO and CUC. MP includes two elements called, *Flow Statistics Collection and Data Processing*. The first one extracts the statistical information from *Statistics Module* located at CP. The second element is responsible for processing and organizing the information retrieved by *Flow Statistics Collection* to compute CCO and CUC. *Data Processing* also sends the processed information to the RL-agent located at KP while keeps a historical record of the network state into *Data Repository*.

It is noteworthy that MP interacts with CP by a REST-based interface. In turn, MP communicates with KP by specific APIs since, up to now, there are no standardized interfaces for KP.

4.4.4. Data Plane

This plane is responsible for forwarding flows in the monitored SDN by network devices decoupled from CP [106]. Each network device consists of a physical part and a functional part. The physical part comprises hardware elements, such as ports, storage, processor, and memory. The functional part comprises a limited set of operations, such as packet header parsing and extraction of a header field tuple, support for a fixed set of packet operations

(such as header field manipulation and forwarding through a specific set of ports), and the ability to match packet header tuples against a lookup memory primitive (*e.g.*, a hash table or a content addressable memory).

It is important to mention that, in SDN, it is key to perform intelligent monitoring decisions, by MP, CP, and KP, aiming at improving the network performance [28, 39, 2].

4.5. Probing Algorithm

4.5.1. Assumptions

4.5.1.1. Reward

It defines the objective of any RL-agent. In IPro, the reward targets to ensure the accomplishing of network policies regarding CCO and CUC. Let us consider that the RL-agent chooses an action and increases the probing interval. If IPro does not meet with one or more policies (*e.g.,* policy 1 or policy 2), the RL-agent will learn that it is a wrong action; resulting in a negative reward. Conversely, if IPro meets the policies, the RL-agent will learn that such an increase is a good action; resulting in a positive reward.

One of the main RL challenges is to define the reward function. In some cases, the choice of this function is easy because it is implicit in the task. For instance, in an Atari game, there is a score function that is part of the game itself. In other cases, like in IPro, the choice of such function is complex. The RL-agent has a task objective with multiple criteria, such as keeping CCO and CUC within target values to minimize network performance degradation. In the proposed approach, these criteria are combined in a single scalar-valued reward function using the normal distribution defined by Matignon *et al.* [107], whose heuristic allows defining a multi-objective model useful to consider CCO and CUC (control policies) simultaneously. Therefore, the reward function is defined as follows.

$$R\left(S_t, A_t\right) = \beta_c e^{-\frac{d(C(\Theta), C(\Theta)*)^2}{2\sigma_c^2}} + \beta_u e^{-\frac{d(Us, Us*)^2}{2\sigma_u^2}} \tag{4-4}$$

where, β adjusts the amplitude of the function and σ, the standard deviation, specifies the reward gradient influence area. d is the Manhattan distance between the new state s and goal state $s*$. Θ is the set of active flows in the network. $C\left(\Theta\right)$ and $C\left(\Theta\right)*$ are the CCO (cf. equation 4-6) used for statistics collection of the set of flows Θ from the switches in states s and $s*$, respectively. Us and $Us*$ are CUC in states s and $s*$, respectively.

4.5.1.2. Space of Actions

The space of actions affects the future state of the network (e.g., the next CCO and CUC), changing the options and opportunities available to the RL-agent at later times. The effects of actions cannot be fully predicted; thus, the RL-agent must monitor the network frequently and react appropriately (i.e., search process). For example, it must watch the probing interval to avoid the breach of policies.

Algorithm 2: Space of Actions

```
1 if action = increase then
2 |   Increases the current probing interval
3 else if action = reduce then
4 |   Decreases the current probing interval
5 else
6 |   Keeps the current probing interval
7 end
```

The RL-agent performs the search process using the action functions (increase, reduce, and keep) in Algorithm 2 to change the probing interval in each iteration and uses the reward function as the guide to the goal state.

4.5.1.3. Space of States

The space of states represents a signal transferring to the agent some sense of "how the network is" at a particular time. This space is represented as follows:

$$S \equiv f\left(i, l, cpu\right) \tag{4-5}$$

Each $S_t = (i_t, l_t, cpu_t) \in S$ is characterized by the probing interval i_t, CCO l_t, and CUC cpu_t in the time t. CCO corresponds to the bandwidth consumed by IPro when transmitting and receiving Read-State messages between CP and DP. CUC defines the number of instructions carried out in CP because of IPro tasks (*e.g.*, RL-agent execution, processing of data collection messages). The probing interval indicates how often CP must send Read-State Request messages to retrieve flow statistics from switches in DP.

4.5.1.4. Statistics Collection

In SDN, the pull-based monitoring is handled by the controller that interacts with the switches via a control channel over TCP. There are two interaction methods [100] [108]: Per-Flow Collection (PFC) and Per-Switch Collection (PSC).

- In PFC, the controller sends a request to a switch to collect the traffic statistics of one particular flow. This method generates a high CCO when the controller requires to collect statistics of many flows. This overhead is due to the large quantity of Read-State Request messages sent per switch.
- In PSC, the controller sends a request to collect the traffic statistics of all flow entries from a switch. This method reduces the number of Read-State Reply messages (Controller$<-->$Switch) and, so, reduces CCO and CUC. Nonetheless, if PSC is used excessively with, for instance, a low probing interval, it can cause flow statistics overlapping, high CCO, and high CUC.

This master dissertation focuses on the PSC method because first, PSC generates a smaller amount of Read-State messages that imply lower CCO and CUC. Second, PSC reduces the repeated headers of the flows that involve less redundancy in the collected information [108].

In IPro, the **SDN Model** consists of a logically-centralized controller (may be a cluster of distributed controllers [22] [109]) and a set of switches. The SDN is modeled by an out-of-band control plane and an undirected graph $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes (switches and controllers) and $E = \{e_1, ..., e_u\}$ is the set of links connecting nodes. This master dissertation assumes that the controller knows the existing active flows in the network, denoted by $\Theta = \{\theta_1, \theta_2, ..., \theta_m\}$, with $m = |\Theta|$. Thus, it is also reasonable to assume that the controller knows each flow that passes through each switch v_i, denoted by θ_i, with $i = 1, 2, ..., m$. Therefore, the active flows number in switch v_i is $|\theta_i|$. It is noteworthy that the evaluation of CCO and CUC is critical for any out-of-band CP because, first, the control bandwidth is a limited resource and must be analyzed and optimized. Second, CUC is also a constrained resource that must be used appropriately to avoid the wrong behavior of CP and, so, of the underlying DP.

4.5.1.5. Control Channel Overhead

CCO is the bandwidth cost used for statistics collection of a set of flows from the switches. In IPro, the controller generates this cost when requests and receives to and from the switches the statistics of a set of flows θ_i. According to [101][108], the bandwidth cost caused by θ_i involves two parts: (i) the size of the Read-State Request messages l_{rq} sent to switches; and (ii) the size of the Read-State Reply messages l_{rp} that depends on the number of existing flows $|\theta_i|$ in the flow tables. Thus, the bandwidth cost is defined as follows.

$$C(\Theta) = l_{rq} \cdot |V| + l_{rp} \cdot \sum_{i=1}^{|V|} |\theta_i|, \forall i \, |V| \tag{4-6}$$

4.5.1.6. CPU Usage of the Controller

CUC is the number of instructions generated by execution, calculation, and comparison of raw data in IPro. According to [109], CUC can be estimated through a constant (x) that indicates the number of instructions carried out by CPU to fragment the Read-State Reply messages. Therefore, CUC for analyzing n specific flows from Θ is modeled as a linear function of n.

$$CPU \cong |V| * n\,(ReadStateReply) * x, \forall n \in \Theta \tag{4-7}$$

4.5.1.7. Monitoring Accuracy

MA reflects the difference between the real value of a metric and the measured value by IPro. A smaller difference (error) indicates a higher MA. The error is calculated with the following expression:

$$\%error = \frac{|v_C - v_R|}{v_R} * 100 \tag{4-8}$$

where, v_C is the measured value of the metric being monitored and v_R is the real value (or reference). Therefore, MA is as follows:

$$MA = 100\% - \%error \tag{4-9}$$

4.5.2. Functioning

Algorithm 3 presents the probing interval optimization procedure carried out by IPro. The algorithm inputs are the learning factor α, the discount factor γ (*cf.* Equation 4-2), and the exploration method ε (cf. Equation 4-3). The output is the most-rewarding probing interval according to the current network status.

The probing algorithm has two considerations: *i*) it assumes a null initial condition of the $Q(\mathcal{S}, \mathcal{A})$ before the first update occurs (line 1); and *ii*) it starts its execution from a random state that represents the initial values of probing interval, CCO, and CUC (line 3). After these considerations, the probing interval optimization process begins (line 4). In this process, the RL-agent discovers the reward structure and determines the most-rewarding probing interval by interacting with the network. The proposed algorithm performs the following steps in each iteration (lines 5 to 13):

Algorithm 3: Probing Interval Optimization

Require:
Exploration parameter ε
Discount factor γ
Learning factor α

Result: A probing interval

1 Initialize $Q : Q(\mathcal{S}, \mathcal{A}) = 0, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
2 **while** *not reached stopping criterion* **do**
3 Start in state $S_t \in \mathcal{S}$;
4 **while** S_t *is not terminal* **do**
5 Select A_t from S_t using policy derived from Q using ε-greedy exploration method;
6 $A_t \leftarrow \pi(S_t)$ `// Execute probing action`;
7 Modify the probing interval according to the action A_t;
8 $S_{t+1} \leftarrow T(S_t, A_t)$ `// Receive the new state`;
9 $R_{t+1} \leftarrow R(S_t, A_t)$ `// Calculate reward`;
10 $Q_{t+1}(S_t, A_t) \leftarrow (1-\alpha) \cdot Q_t(S_t, A_t) + \alpha \cdot \left[R_{t+1} + \gamma \cdot \max_{\mathrm{A}} Q_t(S_{t+1}, A)\right]$ `// Update Q-function`;
11 Send the probing interval modified to Data Repository;
12 $S_t \leftarrow S_{t+1}$ `// Move to the new state`;
13 $t \leftarrow t + 1$ `// Increment and set the number of steps taken`;
14 **end**
15 **end**

- The RL-agent selects a probing action $A_t = \{up, down, equal\}$ from the Q-function using the ε-greedy exploration method that modifies the probing interval (line 7). The possible actions are to increase (*up*), reduce (*down*), or keep (*equal*) the probing interval.

- The RL-agent executes the probing action selected in the previous step (line 6). Since this execution affects the network behavior, the network falls in a new state. MP determines the new state by Equation 4-5, where CCO and CUC are determined using Equation 4-6 and Equation 4-7, respectively. The value of the probing interval is obtained in the step 1. Subsequently, MP sends this new state to the RL-agent.

- The RL-agent receives the new network state from MP (line 8).

- The RL-agent takes such a state to calculate the reward (line 9). In particular, the reward is computed by Equation 4-4.

- Based on the learning factor, discount factor, initial considerations, reward, and new network state, the RL-agent tunes the values of the Q-function according to Equation 4-2 (line 10).

- The RL-agent sends the probing interval to *Data Repository* (line 11).
- The RL-agent moves to the new state (line 12) and moves on to the next iteration $t+1$ (line 13).

The probing interval optimization process is repeated until the agent perceives that the policy does not change. At this moment, the agent gets the most-rewarding probing interval that keeps CCO and CUC within target values aiming at minimizing network performance degradation caused by the monitoring tasks.

4.5.3. Computational Complexity

IPro determines its optimal policy by finding an Optimal Value Function. The Optimal Value Function of a policy is the expected infinite discounted reward that will be gained, at each state, by executing that policy. This value can be computed by the Equation 4-1, where $Q_t(S_t, A) = E\left(\sum_{t=0}^{\infty} \gamma^t R_t\right)$. Once the IPro RL-agent knows the value of each state under the current policy, it considers whether the value could be improved by changing the first action taken. If the value can be improved, the RL-agent changes the policy to take the new action whenever it is in that situation. This step guarantees an improvement in the performance of the policy strictly. When no enhancements are possible, then the policy is optimal.

Since IPro operates by successive approximations of an Optimal Value Function, its computational complexity, per iteration, is quadratic in the number of states (S) and linear in the number of actions (A): $\left(O\left(|A|\,|S|^2\right)\right)$. Furthermore, the number of iterations required to reach the Optimal Value Function is polynomial in the number of states and the magnitude of the highest reward if the discount factor is held constant. In the worst case, the number of iterations grows polynomially in $\frac{1}{1-\gamma}$. Thus, the IPro RL-agent convergence rate slows considerably as the discount factor nearby 1.

4.5.4. IPro Interactions

IPro uses RL to optimize the probing interval regarding CCO and CUC. For higher compression of how the IPro architecture elements interact at run time, the analysis is divided into two general parts: statistics collection process and probing interval optimization process.

4.5.4.1. Statistics Collection Process

Figure **4-5** depicts how the elements of CP and DP interact in the statistics collection process.

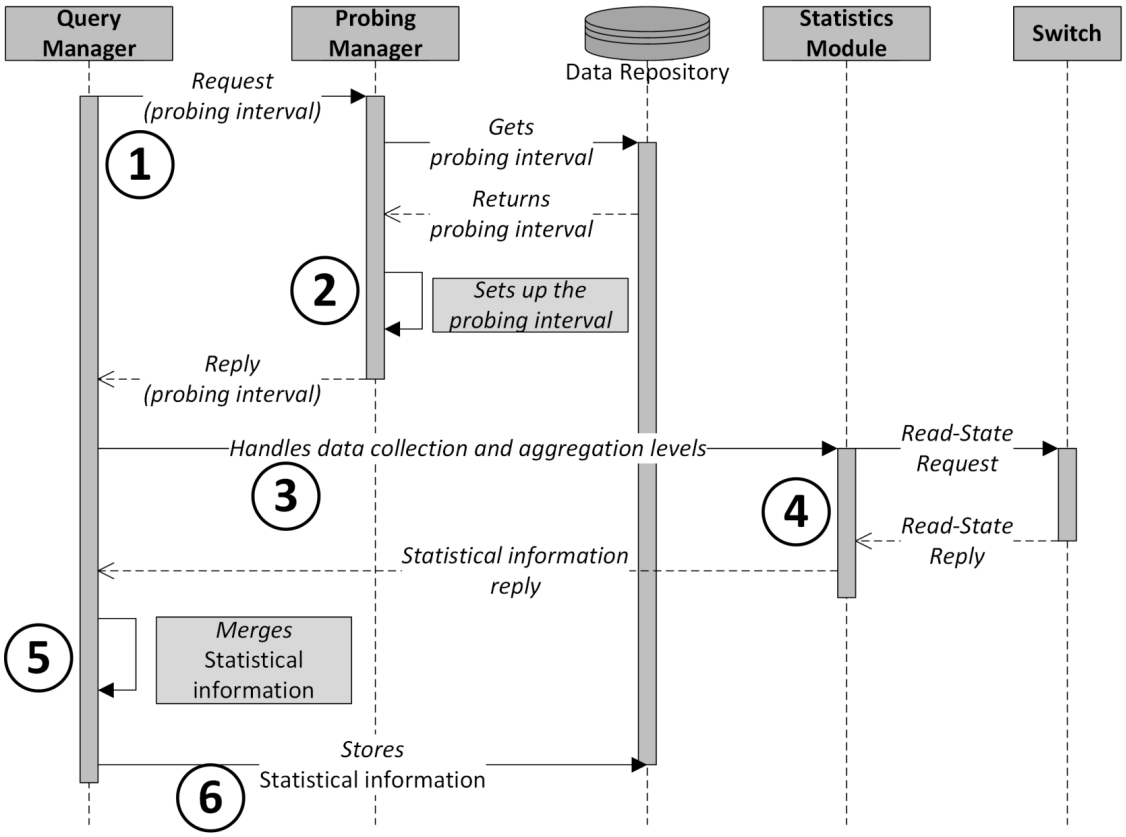

Figure 4-5.: Sequence diagram of statistics collection process

(1) Query Manager requests Probing Manager to set up the probing interval to handle the data collection.

(2) Probing Manager consults the data repository to know the probing interval and sets up it how the current interval.

(3) Query Manager handles the data collection based on the current probing interval and the desired aggregation levels (*e.g.*, byte, packet, flow).

(4) Statistics Module collects statistical information from the switch at the current probing interval using Read-State messages (request and reply). After this data collection, Query Manager merges (5) and stores (6) the statistical information into the Data Repository. Thus, this information can be used ulteriorly by upper-layer applications.

It is important to highlight that the collection process affects network behavior; the network falls in a new state.

4.5.4.2. Probing Interval Optimization Process

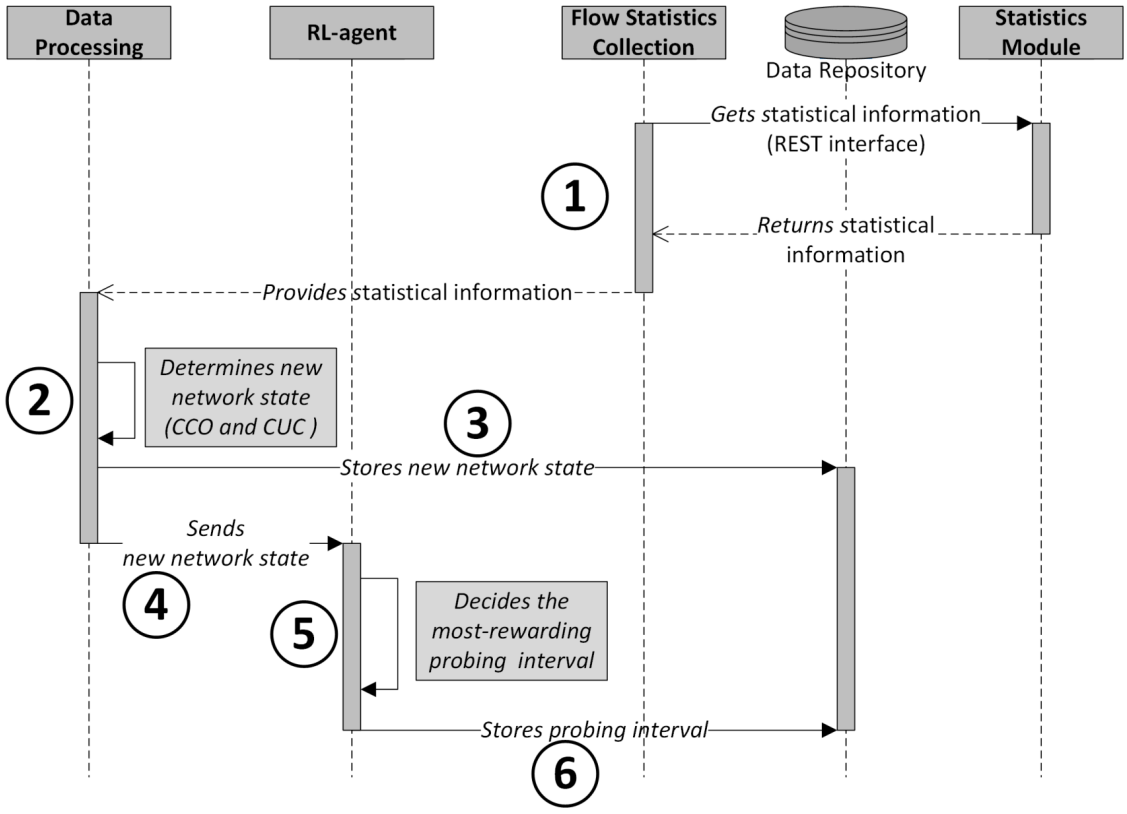

Figure 4-6.: Sequence diagram of probing interval optimization process

Figure **4-6** depicts how the elements of MP and KP interact in the probing interval optimization process.

(1) Flow Statistics Collection extracts the statistical information from the Statistics Module located at CP and provides this information to Data Processing.

(2) Data Processing processes and organizes the information retrieved by Flow Statistics Collection to determine and such a new state by analyzing CCO and CUC.

(3) Data Processing stores the new state.

(4) Data Processing sends the new state to RL-agent located at KP.

(5) The RL-agent takes such a state to calculate the reward. Based on the reward, the RL-agent decides the most-rewarding probing interval intended to minimize CCO and CUC.

(6) The RL-agent stores this new probing interval to Data Repository. Probing Manager applies this interval that affects the network behavior again (*i.e.*, initiates Statistics Collection Process). This process continues until the network administrator decides to stop IPro.

4.6. Final Remarks

Initially, in this chapter, the fundamental concepts about KDN and RL (in particular Q-learning) were described. This master dissertation considered these concepts for proposing a KDN-based architecture (IPro) that keeps CCO and CUC within predefined thresholds and maintains an acceptable MA. Subsequently, the IPro architecture and its architectural elements were introduced and detailed. Finally, the RL-based algorithm (its functioning, computational complexity, and interactions) that determines the probing interval considering network traffic variations, CCO, and CUC was presented.

5. Evaluation

This chapter provides an extensive evaluation of the feasibility of using IPro as a practical approach (*i.e.*, in terms of CCO, CUC, and MA) for intelligent probing in SDN. This chapter starts showing the test environment, including the SDN to monitor. After, this chapter presents the IPro prototype developed and its impact (*i.e.,* the change over the time of Probing Interval) in CCO, CUC, and MA (of the throughput) in the test environment. This chapter finishes showing the comparison of IPro with PPA and other adaptive approaches by quantitative and qualitative analysis, respectively.

5.1. Setup

5.1.1. Test Environment

Figure **5-1** presents the test environment used to evaluate IPro. This environment includes the campus network topology of the Federal University of Rio Grande do Sul [39], the Ryu Controller, and the IPro prototype (*cf.* Section 5.1.2). The monitored topology includes 11 OpenFlow switches (version 1.3) that connect 230 hosts from 7 laboratories (with 20, 30, and 40 hosts) and 4 administration offices (each one with 10 hosts) through links with a bandwidth of 10 Mbps each one. Furthermore, such topology includes a Web Server and a File Server connected in one of the administration offices. Each switch of the network is connected via a dedicated TCP channel to a remote Ryu controller used to coordinate network-wide forwarding decisions. The Mininet emulator [110] was used to deploy the monitored topology, which runs on a Ubuntu server with an Intel i7-4770 2.26 GHz and 8GB RAM. The Ryu Controller runs on a Ubuntu server with a Core i7-4770 processor and 2GB RAM.

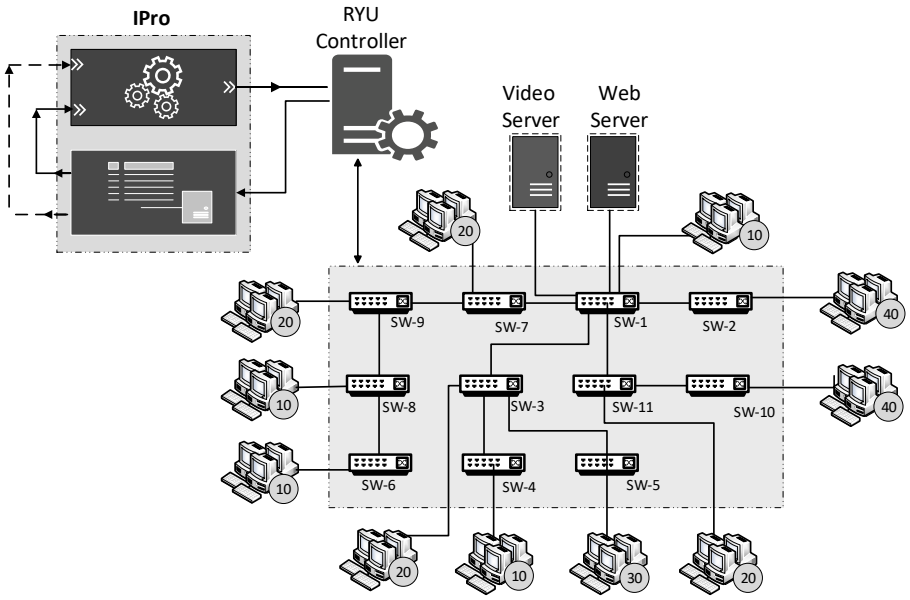

Figure 5-1.: Test Environment

To reliably test IPro, a realistic evaluation framework reflecting current and forecasted traffic patterns (*e.g.,* web, P2P, and video) was used. In this master dissertation, to two measurement studies that investigate the composition of realistic traffic patterns [111, 112] was resorted. The results indicate the dominance of Web traffic, amounting to 52% overall measured traffic, followed by video traffic with 25-40% of all traffic, while P2P traffic constituting only 18.3% of total traffic. Thus, in all experiments, only Video and Web traffic was generated, in a proportion of 75% to 25%, respectively.

The Video traffic was generated by using the VLC media player [113] that can be used as a server and as a client to stream and receive video streams. The Web traffic was generated by using the Apache Server [114] and http-clients based on Linux wget. These servers and clients were included in the campus topology. In this evaluation was assumed that all 230 hosts are active during the whole experiment time and place a request on average every 30 seconds. Furthermore, all experiment results have a confidence level equal to or higher than 95%.

5.1.2. Prototype

Figure **5-2** depicts the IPro prototype including: *RL-agent*, *Data Processing*, *Flow Statistics Collection*, *Data Repository*, *Probing Manager*, *Query Manager*, and *Statistics Module*. *RL-agent* and *Data Processing* were developed and deployed using version 1.15 of Numpy that

is the fundamental Python package for scientific computing. *Probing Manager*, *Query Manager*, and *Statistics Module* were developed using the REST-based API provided by Ryu. This API helps to retrieve the switch statistics. *Flow Statistics Collection* was developed using the Ryu API based on Python. *Data Repository* was developed in MySQL. The IPRo prototype (including all test scripts) is available in [115].

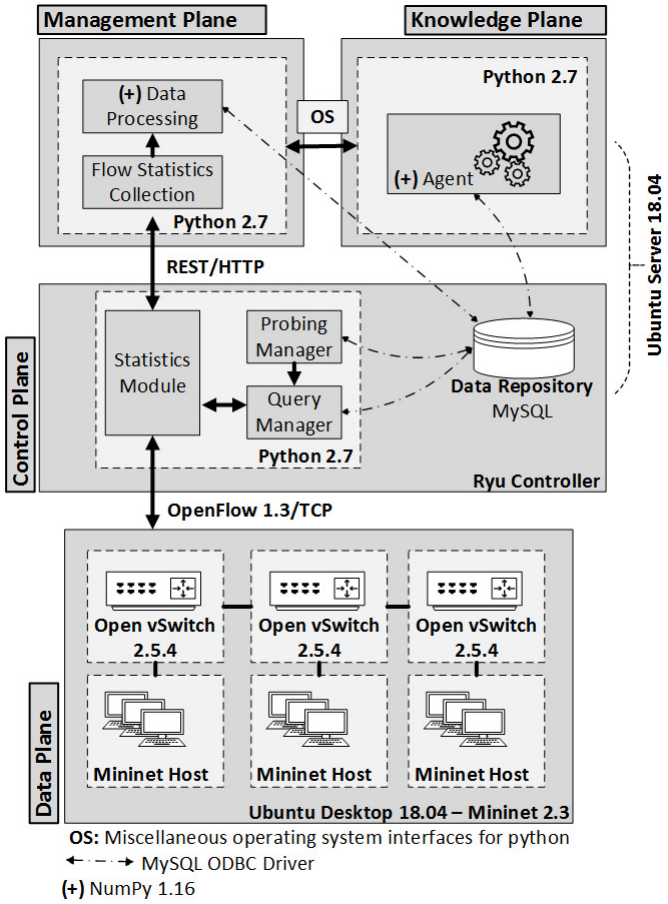

Figure 5-2.: IPro - Prototype

It is important to highlight that the *Statistics Module* interacts with DP by the OpenFlow Protocol [101]. This protocol was used because it has progressively turned on the SBI *de-facto* standard in SDN [29]. OpenFlow describes an open protocol that allows software applications to program (*i.e.*, add, update, and delete flow entries) the flow table of the OpenFlow-compliant switches. In particular, the *Statistics Module* uses the OpenFlow version 1.3. Specifically, this module uses two Read-State (Request-Reply) messages to collect information from the switch, such as current configuration, statistics, and capabilities. The controller sends a Read-State Request message to the switches to request the traffic statistics of flows. The switches communicate to the controller the requested traffic statistics via

Read-State Reply messages. To get OpenFlow details, the reader to Openflow specification is refered [101].

5.1.3. Space of States

To determine the finite space of states (*cf.* Equation 4-5) that the IPro RL-agent needs to operate, first, the CCO (*cf.* Equation 4-6), CUC (*cf.* Equation 4-7), and MA (*cf.* Equation 4-9) were estimated and experimentally measured when the probing interval varies. The probing intervals between 1 and 15 seconds were manually tested (in steps of 1 second). For each interval, the test duration was 600 seconds. Second, the CCO and CUC were discretized by using the obtained results.

5.1.3.1. Control Channel Overhead

Aiming at determining the space of states that the IPro RL-agent needs to operate, Figure **5-3** presents the impact of the probing intervals on CCO. If the network is monitored with a probing interval upper or equal than 5 seconds, the overhead is lower than 12%. When this interval is smaller than 5 seconds the overhead increases (top than 20%) due to the big size and quantity of Read-State (Request-Reply) messages (*cf.* Equation 4-6). These results corroborate that the probing interval affects CCO significantly.

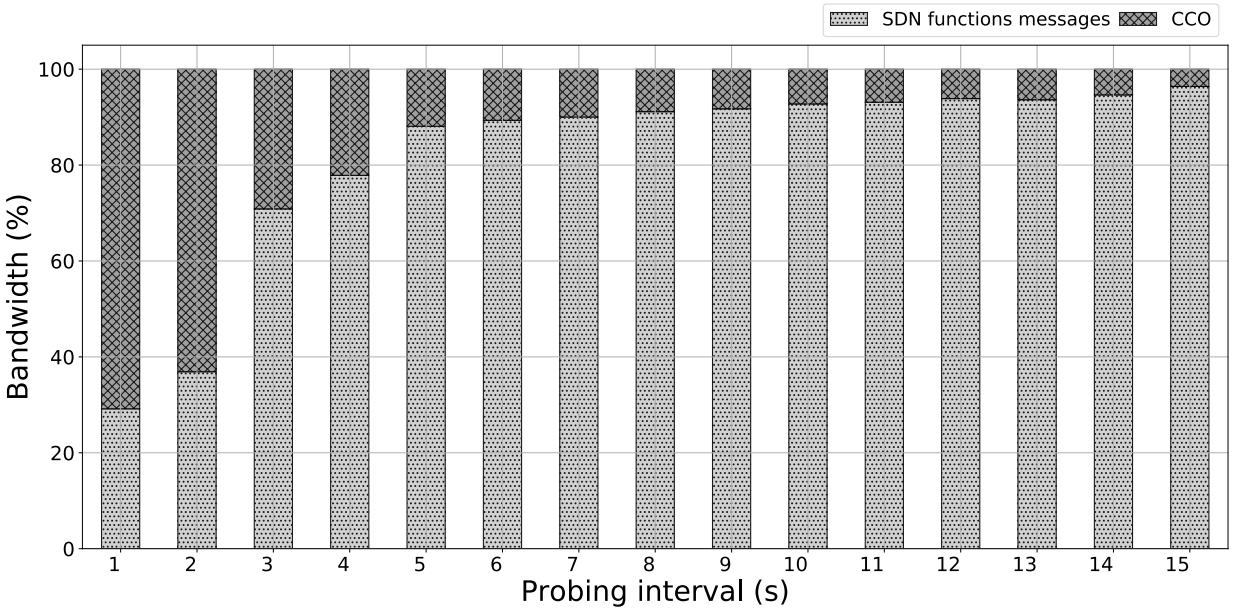

Figure 5-3.: CCO Variation

5.1.3.2. CPU Usage of the Controller

Aiming at determining the space of states that the IPro RL-agent needs to operate, Figure **5-4** presents the impact of the probing intervals on CUC. If the network is monitored with a probing interval upper or equal than 5 seconds, CUC increases on average 8.5% approximately. Nevertheless, when this interval is smaller than 5 seconds, CUC increases (top than 20.5%) because the controller collects statistics information more frequently. As a consequence, the controller CPU must process a higher amount of instructions per second for (de)fragmenting and reading the Read-State messages. Overall, these results corroborate that the probing interval can affect CUC significantly.

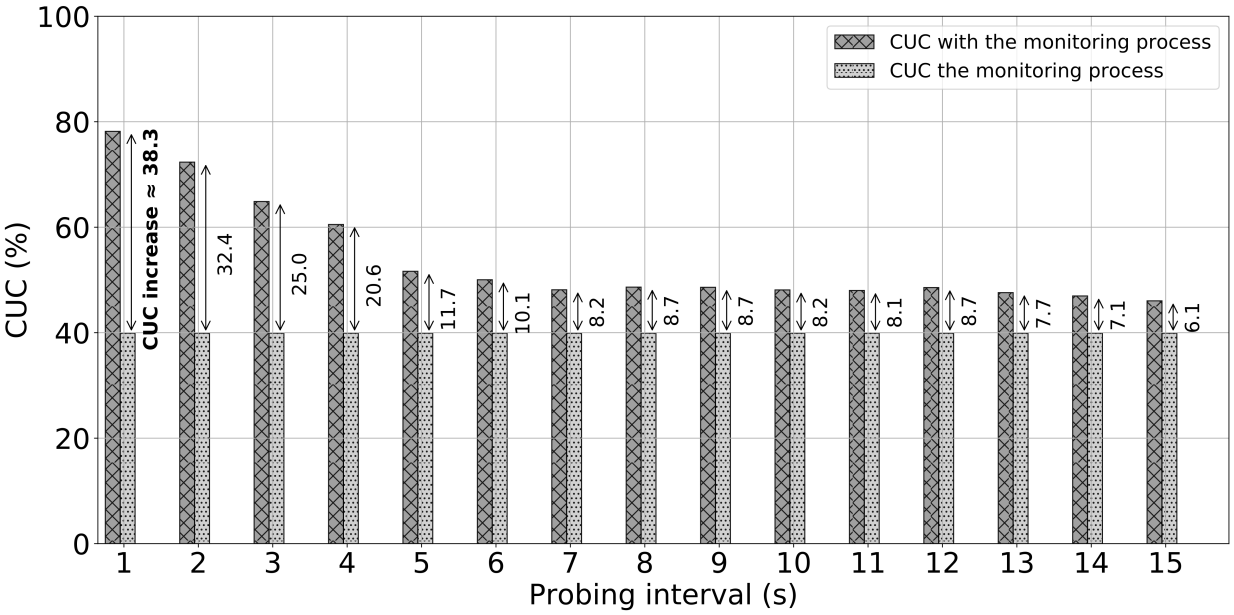

Figure 5-4.: CUC Variation

5.1.3.3. Monitoring Accuracy

To evaluate MA, the throughput metric was measured in each probing interval and determine its respective accuracy using Equation 4-9. Figure **5-5** depicts the evaluation results, disclosing that MA in the throughput measured is higher than 80% when the probing interval varies between 4 and 6 seconds. In particular, the interval of 5 seconds reports the highest MA (88.83%). In turn, the intervals of 6 seconds and 4 seconds achieve an MA equal to 86.06% and 83.83%, respectively. Nonetheless, MA reduces considerably for the other probing intervals. For instance, the interval of 7 seconds accomplishes an MA lower than 69.93% due to the reduction in the quantity of collected information. In turn, the probing intervals 1, 2, and 3 seconds lead to high CCO that interferes with SDN management

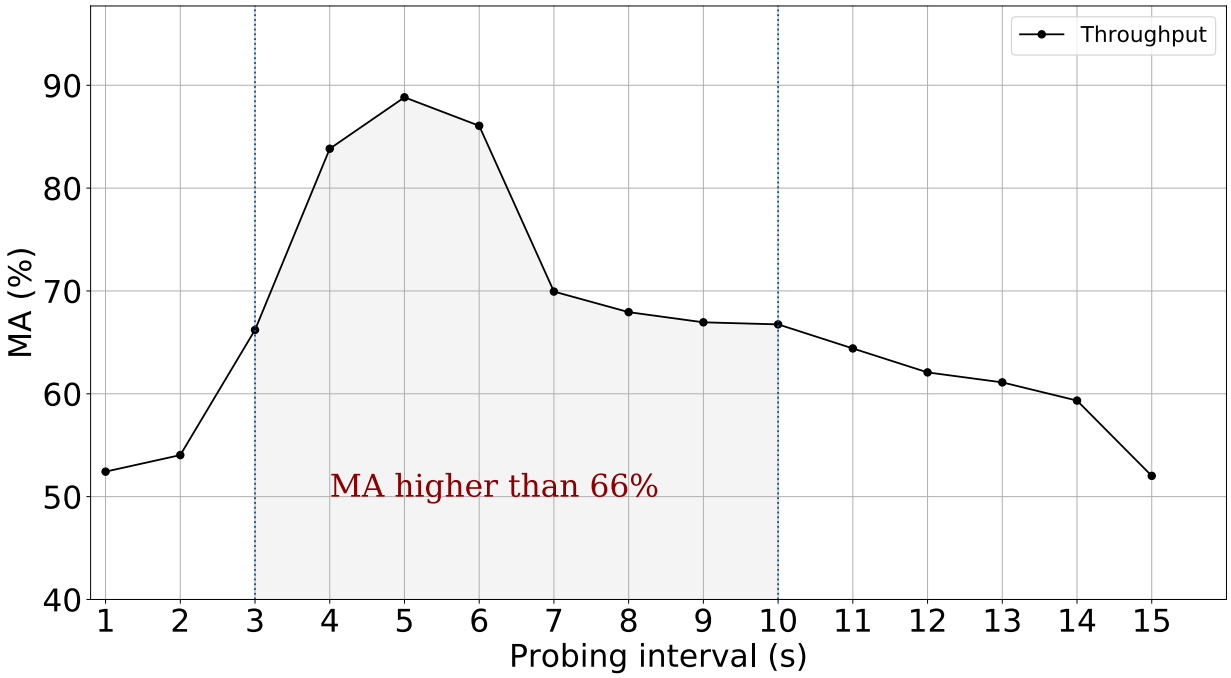

Figure 5-5.: MA of throughput

messages. Furthermore, these intervals also lead to high CUC that compromises the correct operation of the controller and, so, of the underlying data plane. These two facts deteriorate the correct operation of the network generating high TCP errors and low processing of SDN management messages (*cf.* Figure **5-6**). Therefore, the MA of data collected is also affected. In summary, the above results corroborate that the probing interval affects MA significantly.

5.1.3.4. Spaces Discretization

Since IPro is RL-based, it models its environment as a finite MDP. In order to get a finite space of states, the CCO and CUC were discretized by using the results above presented (*cf.* Figures **5-3** and **5-4**) as follows:

1. The values of CCO and CUC are represented in the interval $[0, 1]$, where 0 represents 0% and 1 represents 100%.

2. The policy 1 (related to the threshold ω) is set to 80% aiming at preventing response times of controller upper than 1 millisecond [99]. Thus,
   CCO: $l = \{[0, 0.4), [0.4, 0.5), [0.5, 0.6), [0.6, 0.7), [0.7, 0.8)\}$.

3. The policy 2 (related to the threshold χ) is set to 80% targeting to avoid interference with essential SDN functions (*e.g.*, packet forwarding and route updating) and the

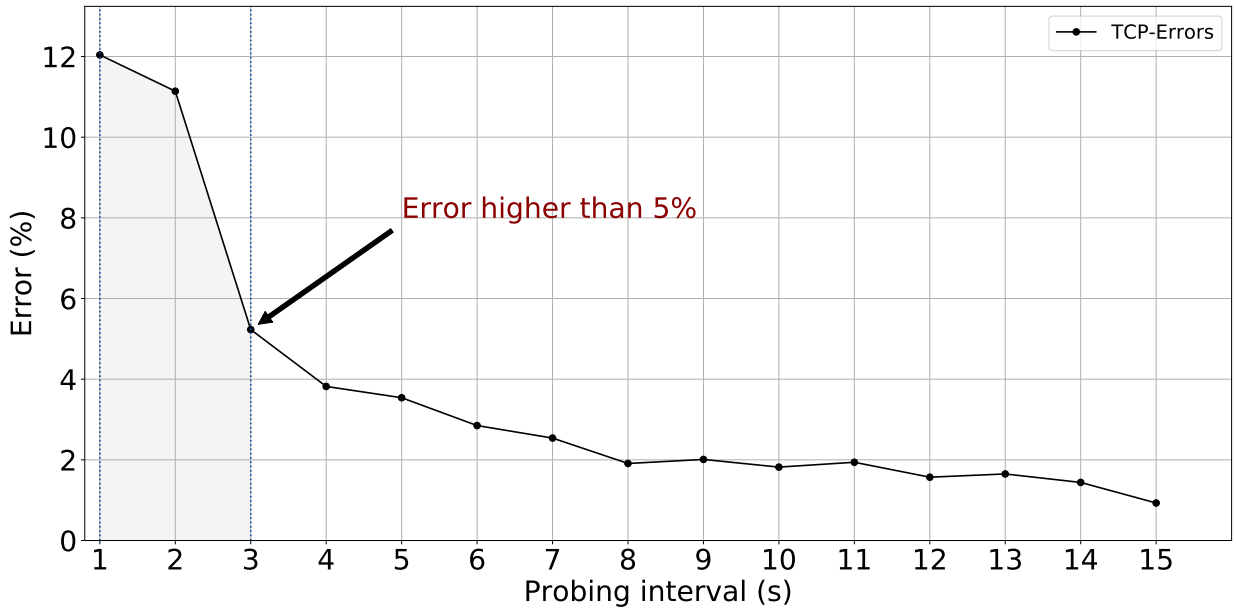

Figure 5-6.: TCP errors generated by Monitoring Interference

reduction of network performance [100]. Thus,
CUC: $cpu = \{[0, 0.4), [0.4, 0.5), [0.5, 0.6), [0.6, 0.7), [0.7, 0.8)\}$.

4. The Probing Interval: $i = \{[4, 10]\}$.

It is important to highlight that CCO and CUC can be divided into smaller sub-intervals to facilitate the RL-agent decision making process. However, smaller intervals increase the size of the space of states and, so, slow down the learning process convergence rate. Thus, the discretized space of states is:

$$S \equiv f(i, l, cpu) : i \in [4, 10], l = \{0, 0.4, 0.5, 0.6, 0.7, 0.8\}, \\ cpu = \{0, 0.4, 0.5, 0.6, 0.7, 0.8\} \tag{5-1}$$

5.2. Intelligent Probing Behavior

To evaluate the IPro prototype (*cf.* Algorithm 3), its impact (*i.e.,* the change over the time of Probing Interval) in CCO, CUC, and MA (of the throughput) was tested in the test environment described in Section 5.1. Figure **5-7** depicts the evaluation results, disclosing diverse facts.

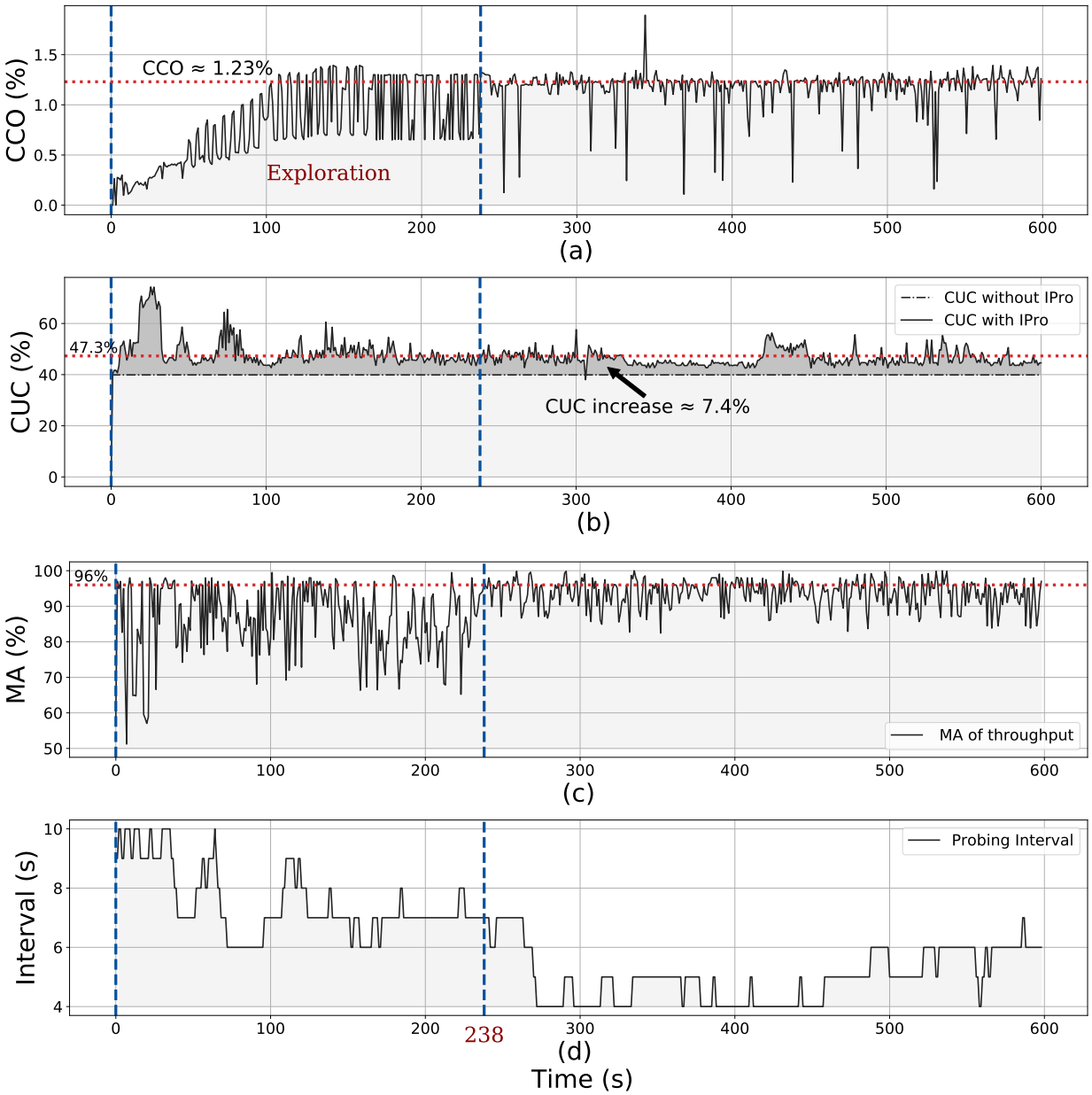

Figure 5-7.: Behavior of the CCO, CUC, MA, and Probing Interval

- During the first 238 seconds (convergence time), CCO, CUC, and MA present a highly fluctuating behaviour Figure **5-7**(a,b,c). This behaviour is because the RL-agent does not have a previous knowledge (*i.e.*, the Q-function starts empty and is filled during this time). Therefore, it is necessary that such an agent starts the exploration process to determine the effect of each action on the network status (*i.e.*, learning process). Figure **5-7**(d) illustrates how the Probing Interval changes during the learning pro-

cess. As the learning process advances, the RL-agent visits each state of the space of states (*cf.* Equation 5-1) multiple times aiming at finding the most-rewarding probing strategy (*i.e.*, the convergence of the learning process).

- After the convergence time, IPro has a CCO near to 1.23%, a CUC around 7.4%, and a MA about 96%. This result demonstrates that, in terms of CCO, CUC, and MA, IPro has good behavior. When the learning process tends to converge, the fluctuations of CCO, CUC, and MA decrease to a smaller radius. This convergence is because a normal distribution function (*cf.* Equation 4-4) was chosen whereby IPro gradually moves to adjacent states to the target state (*i.e.*, calibrates the action-value function). In particular, IPro provides the best behaviour regarding CCO, CUC, and MA when it probes the network with intervals between 4 and 6 seconds.

- It is important to highlight that IPro does not stop its learning because, in real networks, the environment will always be changing and evolving.

To determine the performance of IPro itself, the consumption of CPU and memory of its RL-agent was evaluated. Figure **5-8** depicts the evaluation results, disclosing that this RL-agent does not consume intensively the KP resources, approximately 1% -2% of CPU and 30MBytes. These results demonstrate that, in terms of CPU and memory, IPro RL-agent is efficient.

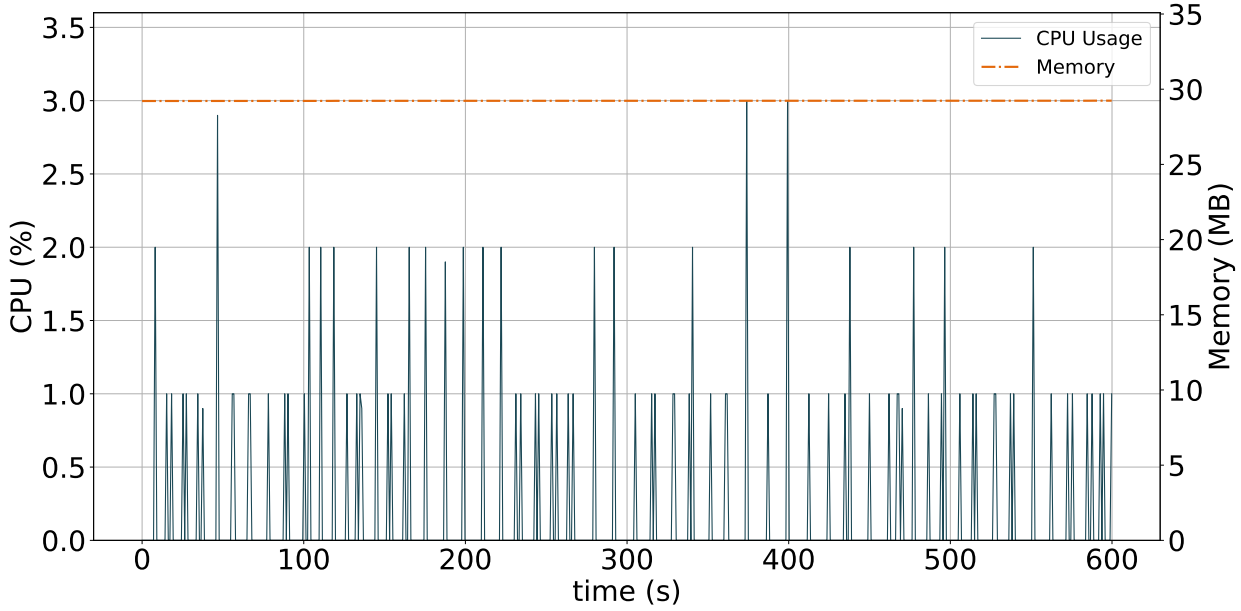

Figure 5-8.: Behavior of the CPU usage and memory of RL-agent

5.3. Comparison

IPro was compared to PPA (*i.e.,* a pull-based approach targeted to monitor the network switches with a pre-defined probing interval) regarding CCO, CUC, and MA after and before the convergence time. The probing intervals tested were ranging between 1 and 15 seconds (in steps of 1 second). For each interval, the test was performed 32 times during 600 seconds (the initial 238 seconds correspond to convergence time), aiming at obtaining the average of CCO, CUC, and MA. Next, only the 4, 5, and 6 seconds probing intervals was analyzed because, in them, the MA of measurements of both throughput and delay metrics is higher than 80%. Finally, the comparison of IPro with other adaptive approaches was extended by a qualitative analysis.

5.3.1. After Converging

Probing Interval [s]	**MA of Throughput [%]**	**MA of Delay [%]**	**CUC [%]**	**CCO [%]**
IPro	96.17	94.78	7.40	1.23
PPA with 4	83.59	82.50	20.60	17.40
PPA with 5	91.38	89.50	11.70	11.45
PPA with 6	86.35	81.03	10.10	11.33

Table 5-1.: Comparison after converging

After converging (*i.e.,* the RL-agent has learned), the experimental results (*cf.* Table **5-1**) reveal diverse facts related to the use of RL for tuning the probing interval:

- IPro has a CCO significantly smaller than PPA. The reduction in the intervals of 4, 5, and 6 seconds is around 16.17%, 10.22%, and 10.1%, respectively.
- IPro uses better the CPU of the controller than PPA about 13.2%, 4.3%, and 2.7%, respectively.
- IPro achieves a higher MA when used to measure the throughput metric than PPA about 12.58%, 4.79%, and 9.82%, respectively.
- IPro achieves a higher MA when used to measure the delay metric than PPA about 12.28%, 5.28%, and 13.75%, respectively.

These facts are because, at each time step, IPro uses the network state for improving its control policies and, then, takes the best action based on the improved policies. These

policies lead to better monitoring regarding CCO and CUC. To sum up, the RL-agent of IPro provides a good behavior in CCO and CUC without compromising MA.

5.3.2. Before Converging

Probing Interval [s]	MA of Throughput [%]	MA of Delay [%]	CUC [%]	CCO [%]
IPro	85.58	84.60	7.56	1.23
PPA with 4	87.05	85.30	22.32	17.40
PPA with 5	90.63	91.50	10.12	11.45
PPA with 6	89.44	86.02	11.08	11.33

Table 5-2.: Comparison before converging

Before converging (*i.e.,* the RL-agent is learning), the experimental results (*cf.* Table **5-2**) reveal diverse facts related to the use of RL for tuning the probing interval.

- IPro achieves a smaller MA in the throughput measurement than PPA, about 1.44%, 5.05%, and 3.86% in the intervals of 4, 5, and 6 seconds, respectively.
- IPro achieves a smaller MA in the delay measurement than PPA, about 0.7%, 6.9%, and 1.42%, respectively.

These facts are because, in this period, IPro explores the effect of each action on the network status (*i.e.,* learning process). To sum up, IPro requires a time to capture the environment model before converging to the optimal policy. Conversely, as presented in Section 5.3.1, when the RL-agent converges to the optimal policy; it gets the most-rewarding probing interval that minimizes the performance degradation (regarding CCO and CUC) caused by monitoring tasks and improving MA (regarding throughput and delay).

5.3.3. Qualitative Analysis

Table **5-3** presents a qualitative comparison between IPro and other adaptive methods, disclosing diverse facts.

- Methods such as [14, 15, 16] obtain accurate measurements using adaptive techniques and threshold-based methods at expenses of increasing CCO and CPU usage (*i.e.*, imbalance between MA and CCO/CPU). IPro offers an RL-based algorithm that obtains

Table 5-3.: Comparison between IPro and other adaptive methods – H → High and L → Low

Work	Adaptive	Accuracy	Overhead	CPU	ML	Description
IPro	✓	H	L	L	✓	An RL-based algorithm is used to get an optimal probing interval to achieve a trade-off between MA, CCO, and CUC
[14]	✓	H	H	H		Adaptive sampling algorithms are used to tune the load level generated by monitoring process
[15]	✓	H	H	H		Thresholds are used to adjust the load level generated by monitoring process
[16]	✓	H	H	H		An adaptive fetching mechanism monitors per-flow metrics, such as throughput, delay, and packet loss
[17]	✓	H	L	L		An adaptive flows statistical collection method is used to adjust the polling intervals
[22]	✓	H	L	L		A small set of matching rules and secondary controllers are used to identify and monitor aggregate flows
[24]	✓	H	L	L		A self-tuning monitoring mechanism is used to automatically adapt its settings based on the traffic dynamism
[25]	✓	H	L	H		Extra modules are included in the switches to distribute the monitoring tasks in a balanced way
[26]	✓	H	L	L		A two layers hierarchy of controllers is described. The lowest layer polls the flow statistic and forwards statistics to the top layer. The highest layer coordinates the controllers located at the lowest level

accurate measurements with CCO and CPU usage negligible (*i.e.,* achieves a trade-off between MA, CCO, and CUC).

- Methods such as [22, 24, 25, 26] obtain accurate measurements with low overhead using distributed controllers. These methods differ significantly from IPro. Whereas goal IPro is to optimize the probing interval, these methods focus on merges the collected statistics by every controller in an only statistic metric.

- None of these adaptive approaches consider ML-bases mechanisms that optimize such a trade-off by learning from the network behavior, causing potential bottlenecks in the control channel, packet/flow loss, and performance drops.

Finally, in this qualitative analysis, it is essential to highlight two facts. The first one, the convergence time is an intrinsic parameter of RL-based approaches. The RL-agent exploits known actions to obtain a reward and explores new ones to make better decisions. This agent tries a variety of actions to progressively favor those that appear to be the best. The exploration and exploitation principles introduce a challenge related to the balance between them, which is known as the exploration-exploitation dilemma. This master dissertation does not address this challenge. Indeed, in the IPro prototype, the e-greedy exploration method was used and did not test another one. Second, the learning time of any RL-agent

depends mainly on the size of the space of states; due to it, this master dissertation used a finite one.

5.4. Final Remarks

This chapter presented the test environment used to evaluate IPro, its prototype, and the evaluation. To reliably evaluate IPro, a realistic evaluation framework was used reflecting current and forecasted traffic patterns (e.g., web, P2P, and video). In all experiments, only video and Web traffic were generated, in a proportion of 75% to 25%, respectively (given that the size of the video requests generates more traffic than web requests. Furthermore, all experiment results have a confidence level equal to or higher than 95%. Such evaluation was carried out in terms of the following metrics: CCO, CUC, and MA.

The evaluation results revealed several facts:

- IPro has a CCO significantly smaller than PPA. The reduction in the intervals of 4, 5, and 6 seconds is around 16.17%, 10.22%, and 10.1%, respectively.
- In the same intervals, IPro uses better the CPU of the controller than PPA about 13.2%, 4.3%, and 2.7%, respectively.
- In the mentioned intervals, IPro achieves a higher MA when used to measure the throughput metric than PPA about 6.28%, 1.28%, and 4.05%, respectively.
- In the analyzed intervals, IPro achieves a higher MA when used to measure the delay metric than PPA about 9.9%, 6.1%, and 9.58%, respectively. These facts are because, at each time step, IPro uses the network state for improving its control policies and, then, takes the best action based on the improved policies. These policies lead to better monitoring regarding CCO and CUC.
- RL-agent does not consume intensively the KP resources, approximately 1% -2% of CPU and 30MBytes. Therefore, it can be stated that the IPro RL-agent is efficient regarding CPU and memory.

To sum up, the evaluation results demonstrate that, in terms of CCO, CUC, and MA, it is feasible to use the proposed approach for monitoring SDN. In this sense, such results confirming the relevance of the concepts of KDN (SDN and RL).

From a qualitative point of view, the main characteristics provided by the proposal introduced in this master dissertation are: first, IPro offers an RL-based algorithm that obtains accurate measurements with CCO and CPU usage negligible (*i.e.,* achieves a trade-off between MA, CCO, and CUC). Second, none of these adaptive approaches consider ML-bases

mechanisms that optimize such a trade-off by learning from the network behavior, causing potential bottlenecks in the control channel, packet/flow loss, and performance drops. Third, current adaptive methods differ significantly from IPro. Whereas goal IPro is to optimize the probing interval, these methods focus on merges the collected statistics by every controller in an only statistic metric.

According to the evaluation results and the qualitative characteristics of the proposed approach, it can be considered as a step forward in the network monitoring. KDN is led to a novel application domain located at the intersection of ML, SDN, and network monitoring.

6. Conclusions

This chapter starts summarizing the research work carried out in this dissertation. Then, the answer is provided for the research question raised to guide the verification of the hypothesis defended in this master dissertation. Afterward, the main contributions achieved when conducting such verification are presented. Finally, directions for future work are outlined.

This master dissertation presented the investigation carried out to verify the hypothesis: **"The Machine Learning allows intelligent probing on SDN, improving the accuracy traffic monitoring and reducing the corresponding overhead"**. Based on the hypothesis, an approach (called IPro) that follows the concept of the KDN paradigm was proposed. Such an approach determines the probing interval that keeps within thresholds (target values) the Control Channel Overhead (CCO) and the Extra CPU Usage of the Controller (CUC) using RL concept.

This dissertation also presented the reference implementation of the IPro by a prototype, as well as its evaluation and analysis. The IPro prototype in an emulated environment by using a campus network topology was evaluated. The evaluation results demonstrated that the proposed approach is effective for network monitoring because it achieved to keep the CCO less than 1.23% and CUC less than 8%. Furthermore, IPro has a better MA ($\geq$ 90%) than the Periodic Probing Approach (PPA), a pull-based approach that monitors the switches with a pre-defined probing interval. The experimental results also indicated that IPro requires considerable time (approximately 238 seconds) to converge to the target state. The contributions achieved in this dissertation are:

- The IPro architecture that provides a simple and efficient solution to monitor SDN-based networks by following KDN.

- The RL-based algorithm that determines the probing interval considering network traffic variations and keeps CCO and CUC within target values.

- The IPro prototype that implements the proposed architecture.

6.1. Answering the Research Question

At the beginning of this dissertation, one research question was defined to guide the investigation about the feasibility of using KDN as a practical approach for intelligent probing in SDN. Such a question is revised and answered in the following paragraph.

Research Question: How to intelligently probing SDN with a high accuracy and with a negligible network overhead?

Answer: TE is an essential tool to assist the SDN operation [7], aiming at improving the utilization of network resources and enhancing the QoS. To carry out TE an efficient and reliable traffic monitoring approach that accurately and timely collects the statistics of flows is necessary. This accurate statistics collection implies an increase of CUC and CCO that can lead to an overload of the controller and significantly interfere with essential SDN functions, respectively. The proposed approach (IPro) permitted to overcome such CUC and CCO, confirming the importance of the concepts of KDN and RL. Using an extensive quantitative evaluation, it is demonstrated that in terms of CCO, CUC, and MA, IPro is an efficient approach for SDN monitoring. In fact, the evaluation results showed that the proposed approach is useful for network monitoring because it achieved to keep the CCO less than 1.23% and CUC less than 8%. Furthermore, IPro has a better MA ($\geq$ 90%) than PPA, a pull-based approach that monitors the switches with a pre-defined probing interval.

6.2. Contributions

This dissertation investigated the feasibility of using KDN as a practical approach for intelligent probing in SDN. The carrying out of such investigation led to the following significant contributions.

- **The KDN-based architecture.** This architecture provides an efficient solution for tuning the probing interval in SDN, which keeps CCO and CUC within predefined thresholds while it maintains an acceptable MA.
- **The RL-based algorithm.** This algorithm determines the probing interval considering network traffic variations, CCO, and CUC.
- **The IPro prototype.** This prototype implements the proposed architecture.

The above-mentioned contributions were reported to the scientific community through paper submissions to renowned journals (see Appendix A).

- A paper published in the journal Computer Networks. Colciencias index: A1.
- A paper submitted to the journal IEEE Latin America Transactions. Colciencias index: A2.

6.3. Future work

During the carrying out of this master dissertation, interesting opportunities for further research were observed. These opportunities are outlined below.

- **Convergence time**. IPro was implemented and analyzed using only Q-learning approach. Therefore, there is an opportunity to extend it by using other methods such as model-free approach (*e.g.,* Q-learning with Experience Replay) and model-based approach (*e.g.,* Deep Reinforcement Learning) to reduce the convergence time.
- **Reward function**. The reward function of IPro was analyzed only in terms of CCO and CUC. Thus, there is a chance to propose its reward function using other parameters such as MA and computational resources of switches, aiming at improving the probing interval estimation.

Bibliography

[1] P. Tsai, C. Tsai, C. Hsu, and C. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, pp. 3958–3969, Dec 2018. xvii, 8

[2] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017. xvii, 22, 23, 28, 31

[3] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt, "Towards sla policy refinement for qos management in software-defined networking," in *IEEE 28th International Conference on Advanced Information Networking and Applications*, pp. 397–404, May 2014. 1, 8

[4] Á. López-Raventós, F. Wilhelmi, S. Barrachina-Muñoz, and B. Bellalta, "Machine learning and software defined networks for high-density wlans," *arXiv preprint arXiv:1804.05534*, 2018. 1

[5] J. d. J. Gil Herrera and J. F. B. Vega, "Network functions virtualization: A survey," *IEEE Latin America Transactions*, vol. 14, pp. 983–997, Feb 2016. 1, 6

[6] F. Estrada-Solano, A. Ordonez, L. Z. Granville, and O. M. C. Rendon, "A framework for sdn integrated management based on a cim model and a vertical management plane," *Computer Communications*, vol. 102, pp. 150 – 164, 2017. 1, 6

[7] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Elsevier, Computer Networks*, vol. 71, pp. 1 – 30, 2014. 1, 7, 56

[8] O. M. C. Rendon, C. R. P. dos Santos, A. S. Jacobs, and L. Z. Granville, "Monitoring virtual nodes using mashups," *Computer Networks*, vol. 64, pp. 55–70, 2014. 1, 9

[9] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, *FlowSense: Monitoring Network Utilization with Zero Measurement Cost*, pp. 31–41. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. 1

[10] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *IEEE 34th International Conference on Distributed Computing Systems*, pp. 228–237, June 2014. 1

[11] M. Aslan and A. Matrawy, "On the impact of network state collection on the performance of sdn applications," *IEEE Communications Letters*, vol. 20, no. 1, pp. 5–8, 2016. 1, 10

[12] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Flowcover: Low-cost flow monitoring scheme in software defined networks," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 1956–1961, IEEE, 2014. 1, 2, 10, 18, 19

[13] K. Dharsee, E. Johnson, and J. Criswell, "A software solution for hardware vulnerabilities," in *2017 IEEE Cybersecurity Development (SecDev)*, pp. 27–33, IEEE, 2017. 1

[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE Network Operations and Management Symposium*, pp. 1–9, May 2014. 2, 15, 16, 18, 19, 51, 52

[15] D. Raumer, L. Schwaighofer, and G. Carle, "Monsamp: A distributed sdn application for qos monitoring," in *Federated Conference on Computer Science and Information Systems*, pp. 961–968, Sept 2014. 2, 15, 18, 19, 51, 52

[16] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *IEEE Network Operations and Management Symposium*, pp. 1–8, May 2014. 2, 15, 18, 19, 51, 52

[17] H. Tahaei, R. Salleh, S. Khan, R. Izard, K.-K. R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Measurement*, vol. 95, pp. 317–327, 2017. 2, 16, 18, 19, 52

[18] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, *OpenTM: Traffic Matrix Estimator for OpenFlow Networks*, pp. 201–210. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. 2, 16, 18, 19

[19] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker, "Hone: Joint host-network traffic management in software-defined networks," *Journal of Network and Systems Management*, vol. 23, pp. 374–399, Apr 2015. 2, 16, 18, 19

[20] X. T. Phan and K. Fukuda, "Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks," *Journal of Information Processing*, vol. 25, pp. 182–190, 2017. 2, 17, 18, 19

[21] L. Liao, V. C. M. Leung, and M. Chen, "An efficient and accurate link latency monitoring method for low-latency software-defined networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, pp. 377–391, Feb 2019. 2, 17, 18, 19

[22] L. Jose and M. Yu, "Online measurement of large traffic aggregates on commodity switches," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Boston, MA, USA,March 29, 2011*, 2011. 2, 17, 18, 19, 21, 33, 52

[23] G. Tangari, D. Tuncer, M. Charalambides, and G. Pavlou, "Decentralized monitoring for large-scale software-defined networks," in *IFIP/IEEE Symposium on Integrated Network and Service Management*, pp. 289–297, May 2017. 2, 8, 17, 18, 19

[24] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, and G. Pavlou, "Self-adaptive decentralized monitoring in software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 15, pp. 1277–1291, Dec 2018. 2, 18, 19, 52

[25] X. T. Phan, I. D. Martinez-Casanueva, and K. Fukuda, "Adaptive and distributed monitoring mechanism in software-defined networks," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, Nov 2017. 2, 17, 18, 19, 52

[26] H. Tahaei, R. B. Salleh, M. F. Ab Razak, K. Ko, and N. B. Anuar, "Cost effective network flow measurement for software defined networks: A distributed controller scenario," *IEEE Access*, vol. 6, pp. 5182–5198, 2018. 2, 16, 18, 19, 52

[27] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–98, Apr. 2014. 6, 7, 15

[28] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan 2015. 6, 31

[29] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634. 6, 43

[30] O. N. Foundation, "Sdn architecture 1.0 overview," *Open Network Foundation*, 2014. 6

[31] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016. 7

[32] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and principles of internet traffic engineering," tech. rep., 2002. 7

[33] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 10, pp. 36–56, First 2008. 7

[34] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2010. 7, 8

[35] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh Conference on emerging Networking Experiments and Technologies*, p. 8, ACM, 2011. 8

[36] V. Mohan, Y. J. Reddy, and K. Kalpana, "Active and passive network measurements: a survey," *International Journal of Computer Science and Information Technologies*, vol. 2, no. 4, pp. 1372–1385, 2011. 8, 9

[37] T. Bujlow, T. Riaz, and J. M. Pedersen, "A method for classification of network traffic based on c5.0 machine learning algorithm," in *2012 International Conference on Computing, Networking and Communications (ICNC)*, pp. 237–241, Jan 2012. 9

[38] S. Dong, D. Zhou, and W. Ding, "The study of network traffic identification based on machine learning algorithm," in *2012 Fourth International Conference on Computational Intelligence and Communication Networks*, pp. 205–208, Nov 2012. 9

[39] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of openflow-based sdn," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 207–215, IEEE, 2015. 9, 31, 41

[40] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 879–894, Aug 2003. 9

[41] B. Claise, "Cisco systems netflow services export version 9," tech. rep., 2004. 9

[42] P. Phaal, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," tech. rep., 2001. 9

[43] B. Claise, "Cisco systems netflow services export version 9," tech. rep., 2004. 9, 15

[44] A. C. Myers and A. C. Myers, "Jflow: Practical mostly-static information flow control," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 228–241, ACM, 1999. 9, 15

[45] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks," in *Distributed Computing Systems. Proceedings. 24th International Conference on*, pp. 628–635, IEEE, 2004. 9, 15

[46] M. Malboubi, Y. Gong, W. Xiong, C. N. Chuah, and P. Sharma, "Software defined network inference with passive x002f;active evolutionary-optimal probing (sniper)," in *24th International Conference on Computer Communication and Networks*, pp. 1–8, Aug 2015. 10

[47] S. Ayoubi, N. Limam, M. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada, and O. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, 2018. 11

[48] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine Learning for Networking: Workflow, Advances and Opportunities," *ArXiv e-prints*, Sept. 2017. 11, 12

[49] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntes-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *arXiv preprint arXiv:1709.07080*, 2017. 11

[50] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012. 11

[51] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006. 11, 12

[52] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, p. 16, Jun 2018. 11, 12, 15

[53] Y. Li, H. Liu, W. Yang, D. Hu, X. Wang, and W. Xu, "Predicting inter-data-center network traffic using elephant flow and sublink information," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 782–792, 2016. 11

[54] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Yanhui Geng, Li Chen, K. Chen, and Hao Jin, "Online flow size prediction for improved network routing," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–6, Nov 2016. 11

[55] Z. Chen, J. Wen, and Y. Geng, "Predicting future traffic using hidden markov models," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–6, IEEE, 2016. 11

[56] B. Bojovic, N. Baldo, and P. Dini, "A cognitive scheme for radio admission control in lte systems," in *2012 3rd International Workshop on Cognitive Information Processing (CIP)*, pp. 1–3, IEEE, 2012. 11

[57] D. Vassis, A. Kampouraki, P. Belsis, and C. Skourlas, "Admission control of video sessions over ad hoc networks using neural classifiers," in *2014 IEEE Military Communications Conference*, pp. 1015–1020, IEEE, 2014. 11

[58] G. Quer, N. Baldo, and M. Zorzi, "Cognitive call admission control for voip over ieee 802.11 using bayesian networks," in *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pp. 1–6, IEEE, 2011. 11

[59] A. Snow, P. Rastogi, and G. Weckman, "Assessing dependability of wireless networks using neural networks," in *MILCOM 2005-2005 IEEE Military Communications Conference*, pp. 2809–2815, IEEE, 2005. 11

[60] X. Lu, H. Wang, R. Zhou, and B. Ge, "Using hessian locally linear embedding for autonomic failure prediction," in *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pp. 772–776, IEEE, 2009. 11

[61] A. Pellegrini, P. Di Sanzo, and D. R. Avresky, "A machine learning-based framework for building application failure prediction models," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pp. 1072–1081, IEEE, 2015. 11

[62] J. Liu, I. Matta, and M. Crovella, "End-to-end inference of loss nature in a hybrid wired/wireless environment," 2003. 11

[63] N. Fonseca and M. Crovella, "Bayesian packet loss detection for tcp," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, pp. 1826–1837, IEEE, 2005. 11

[64] A. Jayaraj, T. Venkatesh, and C. S. R. Murthy, "Loss classification in optical burst switching networks using machine learning techniques: improving the performance of tcp," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 6, pp. 45–54, 2008. 12

[65] E. Castillo, D. C. Corrales, E. Lasso, A. Ledezma, and J. C. Corrales, "Data processing for a water quality detection system on colombian rio piedras basin," in *International Conference on Computational Science and Its Applications*, pp. 665–683, Springer, 2016. 12

[66] E. F. Castillo, W. F. Gonzales, I. D. López, A. Figueroa, D. C. Corrales, M. G. Hoyos, and J. C. Corrales, "Water quality warnings based on cluster analysis in colombian river basins," *Sistemas & Telemática*, vol. 13, no. 33, pp. 9–26, 2015. 12

[67] B. Zhang, I. Valentine, and P. D. Kemp, "A decision tree approach modelling functional group abundance in a pasture ecosystem," *Agriculture, Ecosystems Environment*, vol. 110, no. 3, pp. 279 – 288, 2005. 12

[68] T. Y. Kim, K. J. Oh, I. Sohn, and C. Hwang, "Usefulness of artificial neural networks for early warning system of economic crisis," *Elsevier, Expert Systems with Applications*, vol. 26, no. 4, pp. 583–590, 2004. 12

[69] M. Ø. Nielsen, "Local empirical spectral measure of multivariate processes with long range dependence," *Stochastic Processes and Their Applications*, vol. 109, no. 1, pp. 145–166, 2004. 12

[70] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. Boston, MA, USA: Auerbach Publications, 1st ed., 2011. 12

[71] B. Thuraisingham, "Data mining and cyber security," in *Third International Conference on Quality Software, 2003. Proceedings.*, pp. 2–, Nov 2003. 12

[72] T. O. Ayodele, "Introduction to machine learning," in *New Advances in Machine Learning*, InTech, 2010. 12

[73] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and Semi-supervised Clustering: a Brief Survey," *A Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004. 12

[74] M. Rehman and S. A. Mehdi, "Comparison of density-based clustering algorithms," *Lahore College for Women University, Lahore, Pakistan, University of Management and Technology, Lahore, Pakistan*, 2006. 12

[75] G. Tesauro, "Temporal difference learning and td-gammon," 1995. 13

[76] J. Yoshimoto, S. Ishii, and M.-a. Sato, "Application of reinforcement learning to balancing of acrobot," in *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, vol. 5, pp. 516–521, IEEE, 1999. 13

[77] R. H. Crites and A. G. Barto, "Elevator group control using multiple reinforcement learning agents," *Machine learning*, vol. 33, no. 2-3, pp. 235–262, 1998. 13

[78] K. Doya, "Reinforcement learning in continuous time and space," *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000. 13

[79] S. Schaal and C. G. Atkeson, "Robot juggling: implementation of memory-based learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 57–71, 1994. 13

[80] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, pp. 158–165, Jan 2018. 15

[81] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proceedings IEEE INFOCOM*, pp. 1629–1637, April 2011. 15

[82] Z. Zhang, L. Ma, K. K. Leung, L. Tassiulas, and J. Tucker, "Q-placement: Reinforcement-learning-based service placement in software-defined networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1527–1532, IEEE, 2018. 19

[83] L. S. Sampaio, P. H. Faustini, A. S. Silva, L. Z. Granville, and A. Schaeffer-Filho, "Using nfv and reinforcement learning for anomalies detection and mitigation in sdn," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00432–00437, IEEE, 2018. 19

[84] C. Yu, J. Lan, Z. Guo, and Y. Hu, "Drom: Optimizing the routing in software-defined networks with deep reinforcement learning," *IEEE Access*, vol. 6, pp. 64533–64539, 2018. 19

[85] J. Jiang, L. Hu, P. Hao, R. Sun, J. Hu, and H. Li, "Q-fdba: improving qoe fairness for video streaming," *Multimedia Tools and Applications*, vol. 77, no. 9, pp. 10787–10806, 2018. 19

[86] C. Wang, L. Zhang, Z. Li, and C. Jiang, "Sdcor: Software defined cognitive routing for internet of vehicles," *IEEE Internet of Things Journal*, vol. 5, pp. 3513–3520, Oct 2018. 19

[87] C. H. T. Arteaga, F. Rissoi, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7, IEEE, 2017. 19

[88] P. Megyesi, A. Botta, G. Aceto, A. Pescapé, and S. Molnár, "Challenges and solution for measuring available bandwidth in software defined networks," *Elsevier, Computer Communications*, vol. 99, pp. 48–61, 2017. 21

[89] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 3–10, ACM, 2003. 22

[90] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction- second edition*, vol. 1. Cambridge, Massachusetts, 2018. 23, 24

[91] A. Kolobov, "Planning with markov decision processes: An ai perspective," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–210, 2012. 23

[92] T.-H. Teng, A.-H. Tan, and Y.-S. Tan, "Self-regulating action exploration in reinforcement learning," *Procedia Computer Science*, vol. 13, pp. 18–30, 2012. 24, 25

[93] E. Duryea, M. Ganger, and W. Hu, "Exploring deep reinforcement learning with multi q-learning," *Intelligent Control and Automation*, vol. 7, no. 04, p. 129, 2016. 24

[94] S. Manju and M. Punithavalli, "An analysis of q-learning algorithms with strategies of reward function," 24

[95] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996. 24

[96] C. J. C. H. Watkins, "Learning from delayed rewards," 1989. 24

[97] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Q-learning based congestion-aware routing algorithm for on-chip network," in *Networked Embedded Systems for Enterprise Applications (NESEA), 2011 IEEE 2nd International Conference on*, pp. 1–7, IEEE, 2011. 24

[98] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec 2016. 25

[99] S. Répás, V. Horváth, and G. Lencse, "Stability analysis and performance comparison of three 6to4 relay implementations," 07 2015. 28, 46

[100] H. Xu, Z. Yu, C. Qian, X. Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of sdn using wildcard requests," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, May 2017. 28, 32, 47

[101] O. N. Foundation, "Openflow switch specification v1.3.0," ONF, 2013. 29, 33, 43, 44

[102] A. Doria, J. H. Salim, R. Haas, H. M. Khosravi, W. Wang, L. Dong, R. Gopal, and J. M. Halpern, "Forwarding and control element separation (forces) protocol specification.," 2010. 29

[103] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 127–132, ACM, 2013. 29

[104] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, 2015. 30

[105] S. Denazis, E. Haleplidis, J. H. Salim, O. Koufopavlou, D. Meyer, and K. Pentikousis, "Software-defined networking (sdn): Layers and architecture terminology," 2015. 30

[106] G. Varghese, *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005. 30

[107] L. Matignon, G. Laurent, and N. Le Fort-Piat, "Improving reinforcement learning speed for robot control.," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'06.*, no. sur DVD ROM, pp. 3172–3177, 2006. 31

[108] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Cemon: A cost-effective flow monitoring system in software defined networks," *Computer Networks*, vol. 92, pp. 101–115, 2015. 32, 33

[109] H. Tahaei, R. B. Salleh, M. F. Ab Razak, K. Ko, and N. B. Anuar, "Cost effective network flow measurement for software defined networks: A distributed controller scenario," *IEEE Access*, vol. 6, pp. 5182–5198, 2018. 33, 34

[110] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, p. 19, ACM, 2010. 41

[111] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 75–86, 2011. 42

[112] G. Maier, A. Feldmann, V. Paxson, and M. Allman, "On dominant characteristics of residential broadband internet traffic," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pp. 90–102, ACM, 2009. 42

[113] C. Müller and C. Timmerer, "A vlc media player plugin enabling dynamic adaptive streaming over http," in *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, (New York, NY, USA), pp. 723–726, ACM, 2011. 42

[114] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The apache server," in *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, (New York, NY, USA), pp. 263–272, ACM, 2000. 42

[115] E. F. Castillo, "intelligentProbing prototype and results." accessed November 09, 2019. 43

A. Appendix A - Scientific Production

The research work presented in this master dissertation was reported to the scientific community through paper submissions to renowned journals. The process of doing research, submitting the papers, gathering feedback, and improving the work helped to achieve the maturity hereby presented.

A.1. Papers: accepted and on reviewing

A.1.1. Accepted

Edwin Ferney Castillo, Oscar Mauricio Caicedo Rendon, Armando Ordonez, and Lisandro Zambenedetti Granville. **IPro: An approach for intelligent SDN monitoring**. Computer Networks (COMNET), v. 170, pp. 107108, January 2020, ISSN 1389-1286.

- Type: Journal - Computer Networks
- Status: Published
- Colciencias index: A1
- JSR: Q1

A.1.2. On Revision

Edwin Ferney Castillo, Oscar Mauricio Caicedo Rendon, and Armando Ordonez. **Enabling Adaptive Probing for SDN Monitoring**. IEEE Latin America Transactions (IEEE LATAM).

- Type: Journal - IEEE Latin America Transactions
- Status: Submitted
- Colciencias index: A2
- JSR: Q2

B. Appendix B - Scripts Developed

The scripts developed and presented in this dissertation was reported to the scientific community through GitHub.

B.1. Intelligent Probing Repository

https://github.com/efcastillo7/intelligentProbing