Computer Networks 170 (2020) 107108

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

IPro: An approach for intelligent SDN monitoring

Edwin Ferney Castillo[a,*], Oscar Mauricio Caicedo Rendon[a,*], Armando Ordonez[a], Lisandro Zambenedetti Granville[b]

[a] Department of Telematics, Universidad del Cauca, St 5# 4–70, Popayán, Cauca, Colombia
[b] Institute of Informatics, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, Porto Alegre 9500, RS, Brazil

ARTICLE INFO

Article history:
Received 16 April 2019
Revised 30 December 2019
Accepted 14 January 2020
Available online 14 January 2020

Keywords:
Knowledge-defined networking
Machine learning
Probing interval
Software-defined networking
Traffic monitoring

ABSTRACT

Traffic Monitoring assists in achieving network stability by observing and quantifying its 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 some disadvantages. First, they are unconcerned about the trade-off between probing interval and Monitoring Accuracy (MA). Second, they lack intelligent mechanisms intended to optimize this trade-off by learning from network behavior. This paper introduces an approach, called IPro, to address these shortcomings. Our approach comprises an architecture based on 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 Control Channel Overhead (CCO) and the Extra CPU Usage of the Controller (CUC) within thresholds. An extensive quantitative evaluation corroborates that IPro is an efficient approach for SDN Monitoring regarding CCO, CCU, and MA.

1. Introduction

Software-Defined Networking (SDN) is a paradigm that promotes a flexible architecture for fast and easy configuration of network devices [1]. 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) [2]. Nonetheless, such features are not enough to guarantee an appropriate network behavior when traffic reach unexpected levels [3,4]. In this sense, Traffic Engineering (TE) is an important tool to assist the SDN operation [5]. 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 [6].

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 [7,8], 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) [9] and an Extra CPU Usage of the Controller (CUC) [10]. Third, the additional hardware and software elements can raise security issues [11].

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, in this paper, we 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 [10,12–24]. In particular, [12–16] reduce CCO by using adaptive techniques, wildcards, threshold-based methods, and routing information at expenses of decreasing the Monitoring Accuracy (MA).

* Corresponding authors.
E-mail addresses: efcastillo@unicauca.edu.co (E.F. Castillo), omcaicedo@unicauca.edu.co (O.M.C. Rendon), jaordonez@unicauca.edu.co (A. Ordonez), granville@inf.ufrgs.br (L. Zambenedetti Granville).

https://doi.org/10.1016/j.comnet.2020.107108
1389-1286/

Table 1
Acronyms list.

Acronym	Term	Acronym	Term
AHC	Adaptive Heuristic Critic	ML	Machine Learning
COC	Control Channel Overhead	MP	Management Plane
CP	Control Plane	NM	Network Management
CUC	CPU Usage of the Controller	PFC	Per-Flow Collection
DP	Data Plane	PSC	Per-Switch Collection
InP	Infrastructure Provider	PPA	Periodic Probing Approach
IPro	Intelligent Probing	RL	Reinforcement Learning
ISP	Internet Services Provider	QoS	Quality of Service
KP	Knowledge Plane	SBI	SouthBound Interfaces
MA	Monitoring Accuracy	SDN	Software-Defined Networking
KDN	Knowledge-Defined Networking	SLA	Service Level Agreement
MDP	Markov Decision Process	TE	Traffic Engineering

Other approaches diminish CCO by adding modules or modifying flow tables in the switches [10,17–19] and by adding distributed controllers [20–24]. Thus, these works reduce CCO but may 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.

To overcome these shortcomings, we propose an approach called IPro. Our approach includes an architecture that follows the Knowledge-Defined Networking (KDN) [25] paradigm, an algorithm based on Reinforcement Learning (RL), and an IPro prototype. We use KDN to apply Machine Learning (ML) in SDN. In particular, RL allows optimizing the probing interval of IPro aiming at keeping CCO and CUC within acceptable thresholds. We evaluate the IPro prototype in an emulated environment by using a campus network topology. The evaluation results reveal that IPro keeps CCO less than 1.23% and CUC less than 8%. Furthermore, IPro has better MA ($\geq$ 90%) than the Periodic Probing Approach (PPA), a pull-based approach that monitors the switches with a pre-defined probing interval [26].

To sum up, the main contributions of this paper are:

- A KDN-based architecture that provides an efficient solution for tuning the probing interval in SDN. This tuning aims at keeping CCO and CUC within predefined thresholds, and maintaining an acceptabe 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 remainder of this paper is organized as follows. In section II, we present the background and related work. In section III, we introduce IPro. In section IV, we describe and discuss the case study raised to evaluate IPro. Finally, in section V, we provide some conclusions and implications for future work. For the sake of clarity, Table 1 presents the acronyms used in this paper.

2. Background and related work

In this section, we introduce KDN, RL, and Q-learning briefly. We also present the related work on SDN monitoring.

2.1. Background

KDN encourages the use of ML to change the way the network operators handle, optimize and troubleshoot SDN [25] that breaks the coupling between the control plane and the data plane in the traditional network architecture [1]. In KDN, SDN offers a holistic network view that facilitates centralized control of network policies. Furthermore, SDN provides a flexible architecture that facilitates the performing of complex tasks, such as TE [5], network optimization [27], orchestration [28], and monitoring [29]. In turn, in KDN, ML techniques allow the SDN controller to learn from the network behavior to provide automation (recognize-act) and recommendations (recognize-explain-suggest) targeted to improve the operation and management of network as a whole [30,31].

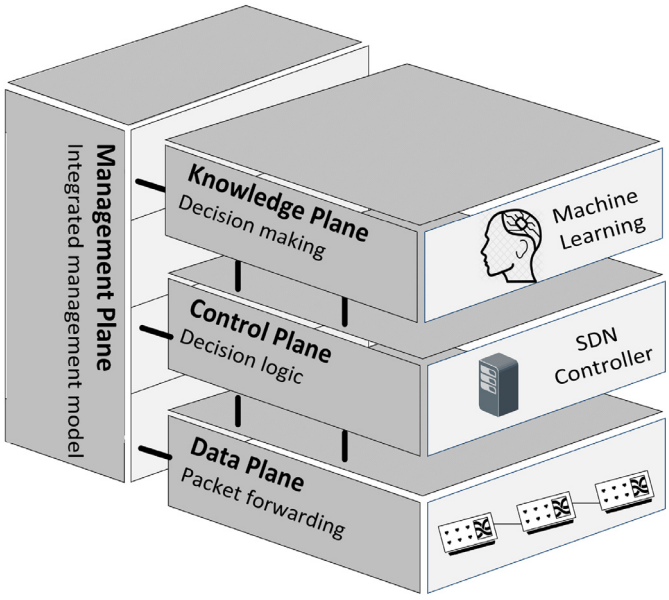


Fig. 1. KDN architecture.

Fig. 1 shows the KDN architecture including its four functional planes. The planes of control, data, and management are inherited from the traditional SDN architecture. The knowledge plane is taken from the David's Clark ideas [32]. The Control Plane (CP) translates the requirements from the applications at network policies and enforces them over network elements by a South Bound Interface (SBI). CP contains one or more controllers (*e.g.*, Floodlight, NOX, POX, and Ryu) that handle and coordinate the network devices located at the Data Plane (DP). DP contains a set of programmable network devices responsible for storing, forwarding and processing data packets. The Management Plane (MP) contains one or more applications (*e.g.*, network monitoring, performance management, configuring/planning resources, and enforcing both policies and contracts) responsible for managing and coordinating each plane individually. DP depends on planes of CP and MP to populate the forwarding tables and update their configuration. The Knowledge Plane (KP) leverages MP and CP to obtain a rich view and control over the network. KP uses ML techniques to extract useful knowledge from the network and

uses that knowledge to make decisions about resource planning, network optimization, performance management, and verification.

The possibilities that emerge from the use of ML in SDN context are various. The TE with traffic classification [33,34], and load balancing [35], DDoS attack detection [36,37], QoE prediction [38], and detection of elephant flows [39]. ML can be divided into three categories according to how the learning is achieved [40,41]: Supervised, Unsupervised, and RL.

Supervised Learning requires a set of instances, commonly known as training data, which are used to define the behavior of ML algorithm. The training data consists of a set of attributes and a class) that will 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 kind of supervised learning algorithms are: Bayesian Networks, Support Vector Machines, k-Nearest Neighbors, Decision Trees, and Neural Networks.

Unsupervised Learning 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 algorithms* aim to obtain a hierarchy of clusters, called dendrogram, that shows how the clusters are related to each other [42].
- *Density algorithms* find the areas with the highest data density, leaving aside the sparsely populated regions, which are considered border points or noise [43].
- *Clustering partitional algorithms* aim to directly obtain a single partition of the collection of items into clusters [42].

RL refers to goal-oriented algorithms (called agent), which learn how to attain a complex objective (goal) by interacting with an environment [44]. 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 as board games [45], job-shop scheduling [46], elevator dispatching [47], and motor control tasks [48], either simulated or real [49]. In this paper, we argue that RL is an ML 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).

RL is an iterative process in which the agent learns a decision-making process by interacting with the environment. Formally, in RL, the environment is typically modeled as a finite Markov Decision Processes (MDP) [50] where the agent sends actions and receives outputs (observations and rewards). In a finite MDP, the agent and environment interact at discrete time steps $t = 0, 1, 2, \ldots, N$. 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, $R_{t+1} \in \mathbb{R}$ at step $t \in N$. Then, the total reward that the agent receives over its lifetime for this particular behavior is:

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

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

RL algorithms find an optimal policy Π^*: $S \rightarrow A$ that maximizes the expected cumulative reward for every state, using the exploration methods (*e.g.*, ε-greedy, Boltzmann [44,51]). 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 due to diverse factors [52]. First, this technique was the pioneering RL method used for control purposes. Second, Q-learning has a learning curve that tends to converge quickly. Third, 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). Fourth, 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 [53,54]. These strategies are out of the scope of this paper.

Q-learning [55,56] 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:

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

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 $\left[R_{t+1} + \gamma \max_{A} Q_t(S_{t+1}, A) \right]$.

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

[1] A domain is defined as categorical if it is finite (discrete numerical values) and un-ordered

[2] A numeric domain consists of real numbers

one of the most used exploration methods [51,57]. 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:

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

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.

2.2. SDN monitoring

Network monitoring is a hot research field that attracts much attention from the academy and industry [30,58,59]. In this section, we present some approaches to SDN monitoring found in the literature.

Zhiyang et al. proposed a low-cost high-accuracy scheme to collect flow statistics, called FlowCover [10]. This scheme proposes a heuristic algorithm to reduce CCO by monitoring a subset of switches. To select the subset of switches, FlowCover first assigns a weight to each switch proportional to the number of flows passing through it. Then, it selects switches with the highest weight. The switches are then monitored according to their weight until all flows across the network are covered.

Raumer et al. introduced an application for the Quality of Service (QoS) monitoring, called MonSamp [13]. 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 [14] 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 MA, CCO, and CUC) and decreases when flows stabilize (reduces MA, CCO, and CUC).

Tahaei et al. proposed a multi-objective network measurement mechanism [15] to overcome CCO, CUC, and MA in a real-time environment. To strike the trade-off between MA and CCO, this mechanism uses an elastic probing for collecting 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 its previous work. This extension proposed a generic architecture for flow measurement in a data-center network. This architecture can use single or multiple controllers [24]. 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 [16] 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 [12]. 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 CCO and MA. To achieve this balance, Payless relies on OpenTM [16] to select only important switches to monitor. Nonetheless, unlike that OpenTM, PayLess offers an adaptive scheduling algorithm for probing, which 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 [17]. HONE uses software agents placed inside end hosts and a module that interacts with OpenFlow switches. HONE integrates the information of the 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, uses database-like tables to represent the statistical information collected from hosts and network devices. Lazy materialization allows the controller and host agents to use the statistical information collected in multiple management tasks. The second technique offers data parallel streaming operators for programming the data-analysis logic.

Phan and Fukuda proposed a scalable framework called SDN-Mon [18]. 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 handles the monitoring functionality (*i.e.,* analyzes, process and manages the monitoring-related messages) in switches to reduce CCO. Phan et al. [23] 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 [19]. 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 injects probe packets only to a set of switches to minimize CCO. This set of switches are selected using a greedy algorithm. To monitor the latency, the agents, first, force to each probe packet to loop around a link for three times, then, calculate their RTT.

Jose and Yu [20] presented a monitoring framework that uses 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, then the counter of the

Table 2
Traffic Monitoring in SDN – H → High and L → Low.

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

matched highest priority rule is updated. On the other hand, the controllers only read and analyze the relevant counters from this memory 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 [21]. 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 [22]. This framework uses a self-tuning, adaptive monitoring mechanism that automatically adjusts its settings based on the traffic dynamics, which improves MA.

Table 2 summarizes relevant works in the field of SDN monitoring, and reveals several facts. First, several works, such as [10,17–24] 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. Second, other works, such as [12–16] 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. 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. In this paper, we address these shortcomings by following the KDN paradigm. In particular, RL is used in the KP of KDN.

To the best of our knowledge, there are not approaches-based on RL for SDN monitoring. However, in the literature, we found some works that use RL for other network tasks. Zhang et al. proposed Q-placement [60], 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 [61]. Yu et al. proposed an RL-based mechanism to achieve universal and customizable routing optimization [62].

Jiang et al. used RL to provide an end-to-end adaptive HTTP streaming media intelligent transmission architecture [63]. 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 [64]. 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 [65].

3. IPro

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

3.1. Motivating scenario

Let us assume that an Infrastructure Provider (InP) use an SDN to offer services to one or more Internet Services Providers (ISPs).

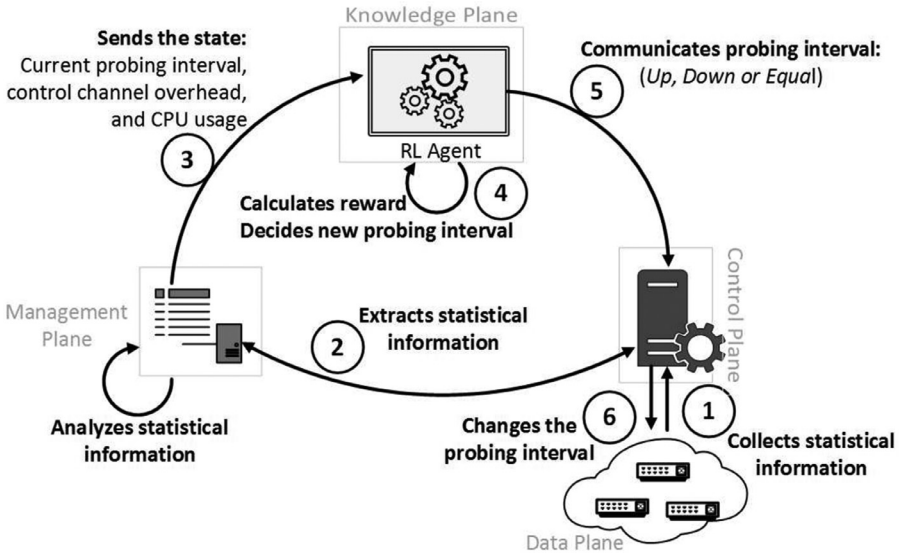


Fig. 2. IPro - High-Level operation.

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 of 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 accurate and timely collects the statistics of flows; these statistics are indispensable to make decisions timely. There are three options to achieve this monitoring. The first one is 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 [20]. The second option is 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 [66]. The third option is 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. In IPro, we offer an adaptive probing that offers such a trade-off by applying RL.

3.2. Overview

IPro applies RL to optimize the probing interval regarding CCO and CUC. Fig. 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. Subsequently, in ③, MP sends this new state to the Knowledge Plane (KP). In 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. In ⑥, CP applies this interval that affects the network behavior again. This operation continues until the network administrator decides to stop IPro.

3.3. Architectural layers and elements

Fig. 3 introduces and details the IPro architecture. IPro has four main planes that follow the KDN fundamentals. Next, we detail these planes.

Knowledge Plane This plane obtains a rich view and global control over the network from MP and CP. Overall, KP operates in three steps. First, it organizes the metadata generated by MP. Second, it converts that metadata into knowledge by using ML techniques. Third, 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 [25]. In this paper, 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 our policy 1 and aims at preventing a high CCO. In turn, χ represents our 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 [67]. In the same sense, when the use of the control channel bandwidth exceeds 80%, the monitoring messages can interfere with the SDN functions [68].

Fig. 4 depicts the general model of our 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 our RL-agent. In the second step, *Experience Accumulation* maps these states and assigns rewards to indicate how good or

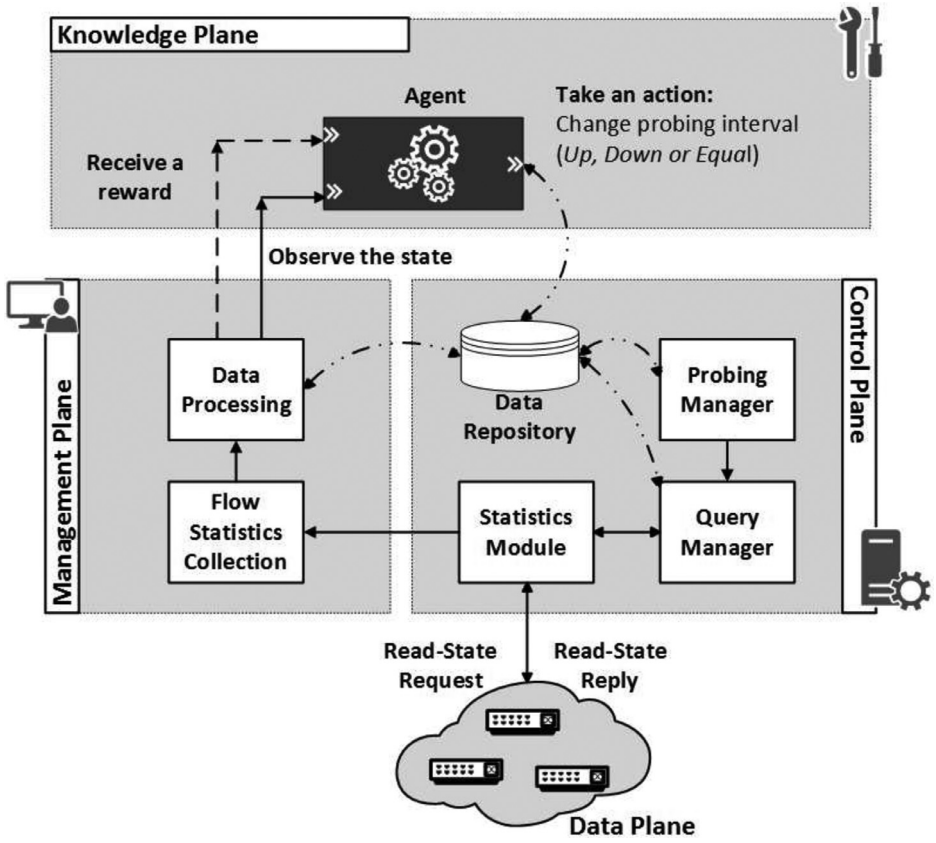


Fig. 3. IPro architecture.

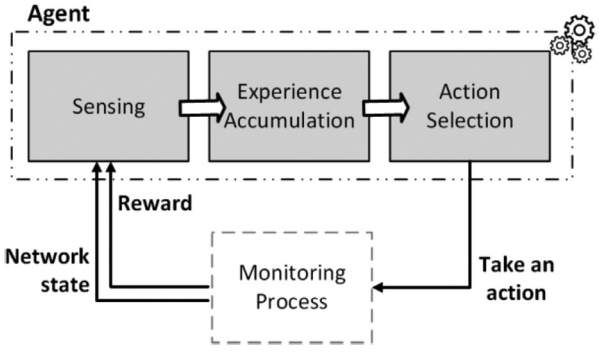


Fig. 4. The RL-agent general model.

Algorithm 1: Data Collection.

Require:
Probing interval I

1 **while** *not reached stopping criterion* **do**
2 **foreach** *switch* $\in$ *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**

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 3.4 details the procedure carried out by the RL-agent.

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 [69], Forwarding and Control Element Separation (ForCES) [70], and Protocol-Oblivious Forwarding (POF) [71]. In this paper, CP handles network monitoring by performing the steps described in Algorithm 1 . First, CP receives decisions about the new probing interval (I) from KP. Second, CP applies these decisions to handle the data collection in the monitored network. Third, CP sends Read-State request messages to each switch connected (line 3) to this plane every I seconds (line 5). Fourth, it receives the Read-State reply messages asynchronously (line 6), which contain the statistical information from the switches. Finally, it 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).

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 this Data Repository.

Management Plane This plane ensures that the network as a whole is running optimally carrying out the Operation, Administration, and Maintenance (OAM) functions [72,73]. 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 paper, 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.

Data Plane This plane is responsible for forwarding flows in the monitored SDN by network devices decoupled from CP. 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 [25,74,75].

3.4. Probing algorithm

3.4.1. Assumptions

Reward 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 our 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), our RL-agent will learn that it is a wrong action; resulting in a negative reward. Conversely, if IPro meets the policies, our 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. Our RL-agent has a task objective with multiple criteria, such as keeping CCO and CUC within target values to minimize network performance degradation. In our approach, these criteria are combined in a single scalar-valued reward function using the normal distribution defined by Matignon et al. [76], whose heuristic allows defining a multi-objective model useful to consider CCO and CUC (control policies) simultaneously. Therefore, we define the reward function as follows.

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

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. $C(\Theta)$ and $C(\Theta)^*$ are the CCO (cf. Eq. (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.

Space of Actions affect 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.

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.

Algorithm 2: Space of Actions.

```
if action = increase then
    Increases the current probing interval
else if action = reduce then
    Decreases the current probing interval
else
    Keeps the current probing interval
end
```

Space of States represents a signal transferring to the agent some sense of "how the network is" at a particular time. We represent the space of states as follows.

$$S \equiv f(i, l, cpu) \quad (5)$$

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.

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 [68,77]: 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. In this paper, we focus 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 [77].

In IPro, the *SDN Model* consists of a logically-centralized controller (may be a cluster of distributed controllers [20,78]) and a set of switches. We model the SDN by an out-of-band control plane and an undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes (switches and controllers) and $E = \{e_1, \ldots, e_u\}$ is the set of links connecting nodes. We assume that the controller knows the existing active flows in the network, denoted by $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$, 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 $i = 1, 2, \ldots, m$. 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 our 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.

Control Channel Overhead 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 [69,77], 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 as follows.

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

CPU Usage of the Controller is the number of instructions generated by execution, calculation, and comparison of raw data in IPro. According to [78], 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.

$$CUC \cong |V| * n(ReadStateReply) * x, \forall n \in \Theta \quad (7)$$

Monitoring Accuracy 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:

$$\%error = \frac{|v_C - v_R|}{v_R} * 100 \quad (8)$$

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:

$$MA = 100\% - \%error \quad (9)$$

3.4.2. 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.* Eq. (2)), and the exploration method ε (cf. Eq. (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):

- 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 Eq. (5), where CCO and CUC are determined using Eq. (6) and Eq. (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 Eq. (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 Eq. (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.

3.4.3. 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 Eq. (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): ($O(|A||S|^2)$). 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.

3.5. 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, we have divided the analysis into two general parts: statistics collection process and probing interval optimization process.

3.5.1. Statistics collection process

Fig. 5 depicts how the elements of CP and DP interact in the statistics collection process. ①, the 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, the 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. Furthermore, it is important to highlight that the collection process affects the network behavior; the network falls in a new state.

3.5.2. Probing interval optimization process

Fig. 6 depicts how the elements of MP and KP interact in the probing interval optimization process. ①, the 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. Data Processing stores the new state ③ and sends it to RL-agent located at KP ④. In 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.

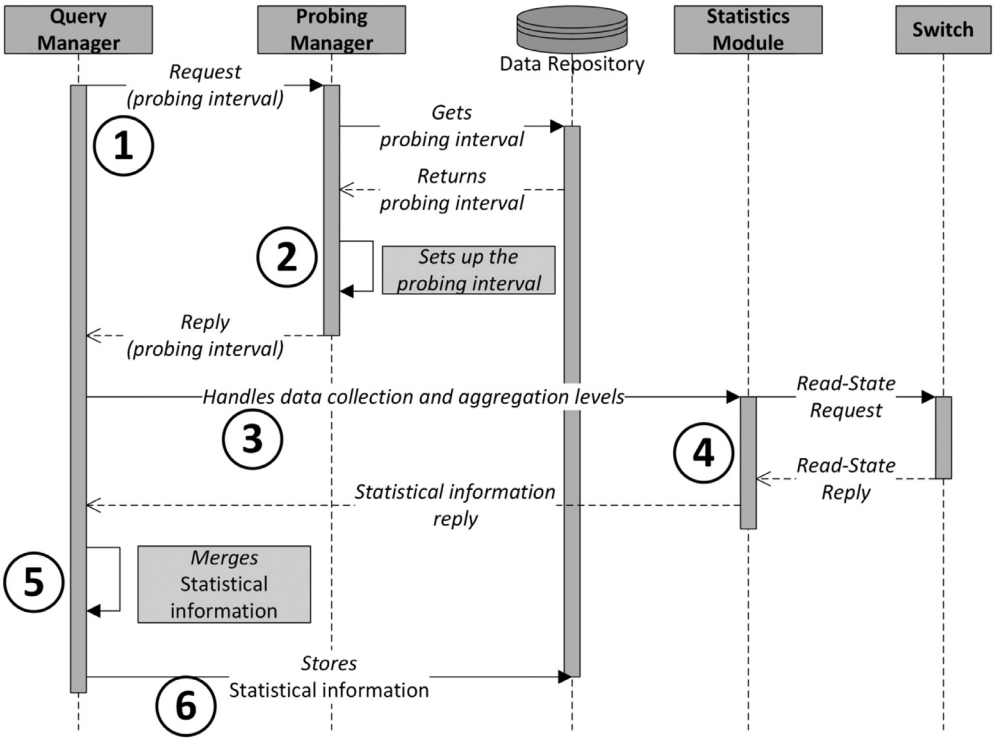


Fig. 5. Sequence diagram of statistics collection process.

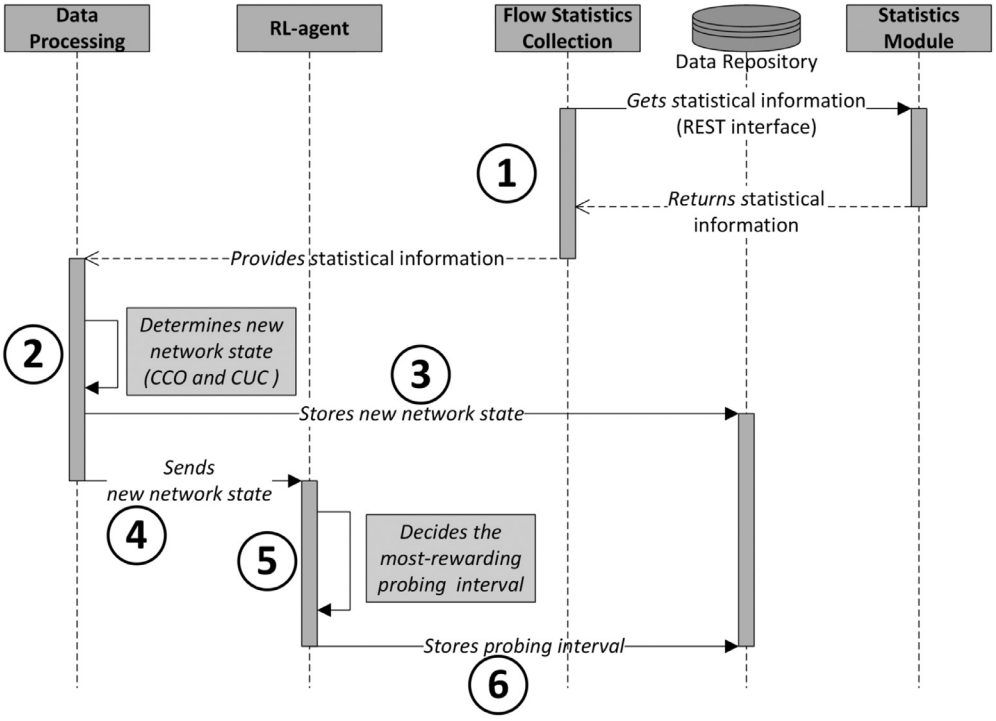


Fig. 6. Sequence diagram of probing interval optimization process.

4. Evaluation

To assess our approach, first, we built a test environment, including the SDN to monitor. Second, we developed an IPro prototype. Third, we conducted experiments to measure the traffic overhead in the control channel, the CUC, and the MA of the prototype built.

4.1. Setup

4.1.1. Test environment

Fig. 7 presents the test environment that we used to evaluate IPro. This environment includes the campus network topology of the Federal University of Rio Grande do Sul [75], the Ryu Controller, and the IPro prototype (*cf.* Section 4.1.2). The monitored

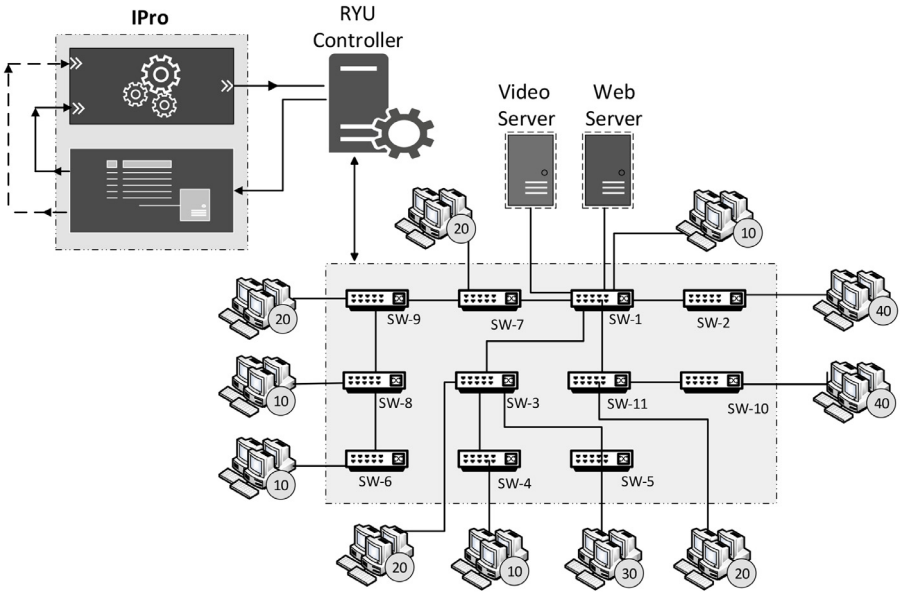


Fig. 7. Test environment.

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 [79] was used to deploy the monitored topology, which runs on a Ubuntu server with an Intel i7-4770 2.26 GHz and 8 GB RAM. The Ryu Controller runs on a Ubuntu server with a Core i7-4770 processor and 2 GB RAM.

To reliably test IPro, we need a realistic evaluation framework reflecting current and forecasted traffic patterns (*e.g.,* web, P2P, and video). In this paper, we have resorted to two measurement studies that investigate the composition of realistic traffic patterns [80,81]. 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, we generated only Video and Web traffic, in a proportion of 75% to 25%, respectively.

The Video traffic was generated by using the VLC media player [82] 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 [83] and http-clients based on Linux wget. These servers and clients were included in the campus topology. We also assume that all 230 hosts are active during the whole experiment time and place a request on average every 30 s. Furthermore, all experiment results have a confidence level equal to or higher than 95%.

4.1.2. Prototype

Fig. 8 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 [84].

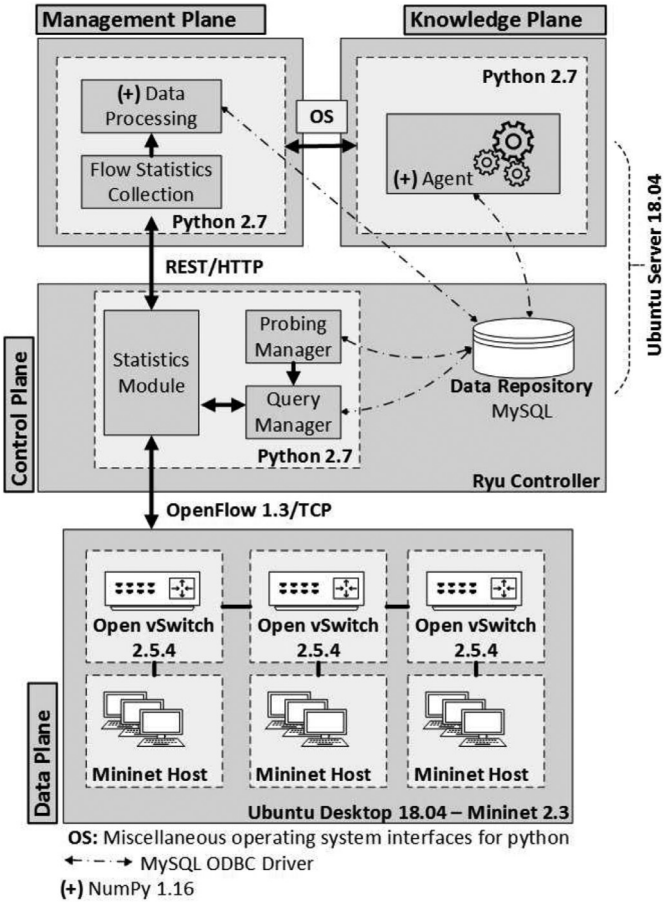


Fig. 8. IPro prototype.

It is important to highlight that the *Statistics Module* interacts with DP by the OpenFlow Protocol [69]. We use this protocol because it has progressively turned on the SBI *de-facto* standard

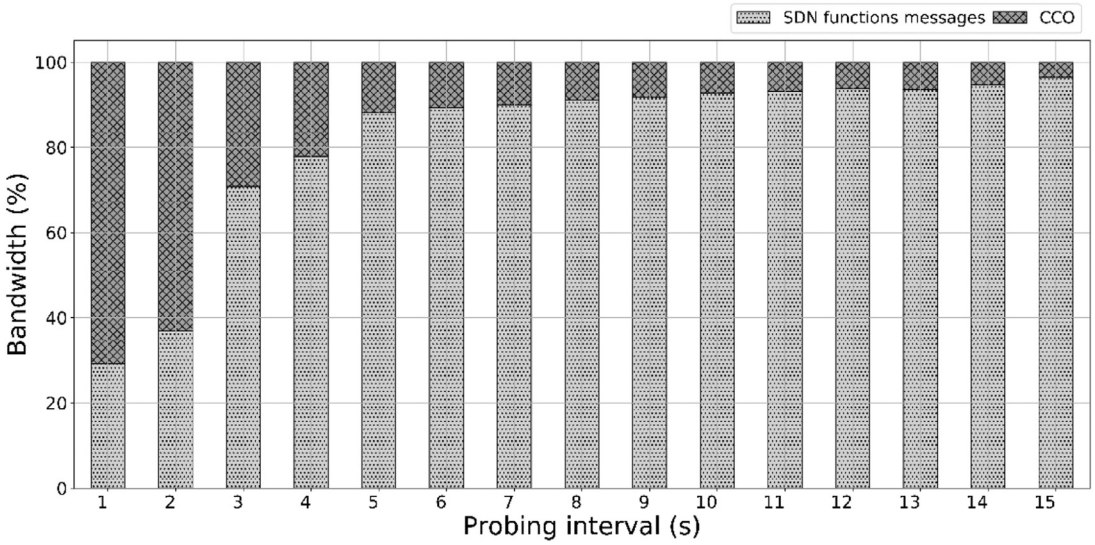


Fig. 9. CCO variation.

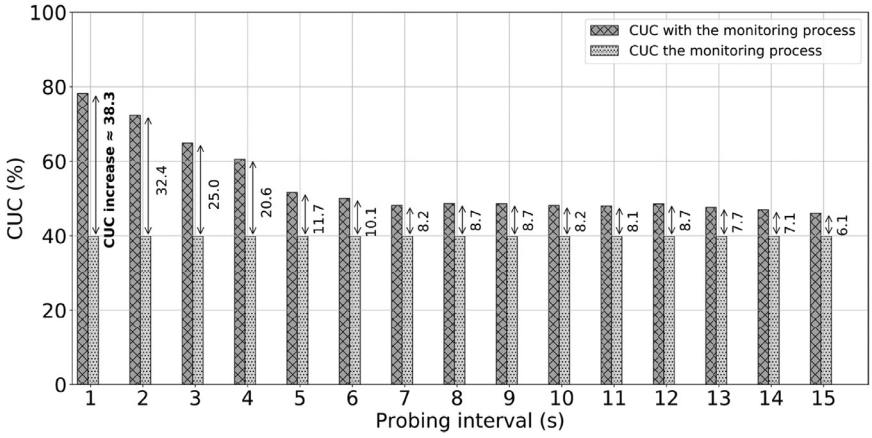


Fig. 10. CUC variation.

in SDN [85]. 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, we refer the reader to Openflow specification [69].

4.1.3. Space of states

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

Control Channel Overhead Aiming at determining the space of states that the IPro RL-agent needs to operate, Fig. 9 presents the impact of the probing intervals on CCO. If the network is monitored with a probing interval upper or equal than 5 s, the overhead is lower than 12%. When this interval is smaller than 5 s the overhead increases (top than 20%) due to the big size and quantity of Read-State (Request-Reply) messages (*cf.* Eq. 6). These results corroborate that the probing interval affects CCO significantly.

CPU Usage of the Controller Aiming at determining the space of states that the IPro RL-agent needs to operate, Fig. 10 presents the impact of the probing intervals on CUC. If the network is monitored with a probing interval upper or equal than 5 s, CUC increases on average 8.5% approximately. Nevertheless, when this interval is smaller than 5 s, 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.

Monitoring Accuracy To evaluate MA, we measure the throughput metric in each probing interval and determine its respective accuracy using Eq. (9). Fig. 11 depicts the evaluation results, disclosing that MA in the throughput measured is higher than 80% when the probing interval varies between 4 and 6 s. In particular, the interval of 5 s reports the highest MA (88.83%). In turn, the intervals of 6 s and 4 s 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 s 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 s lead

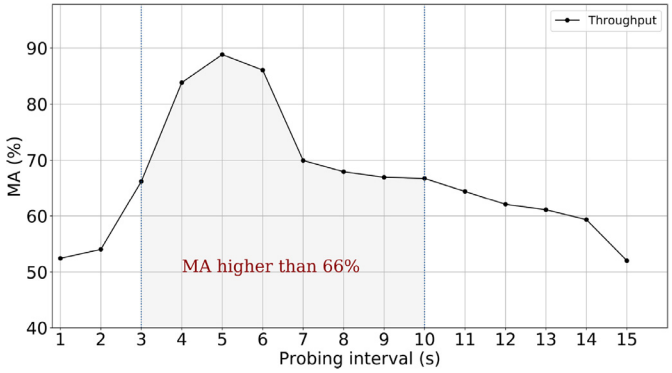


Fig. 11. MA of throughput.

to high CCO that interferes with SDN management 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.* Fig. 12). Therefore, the MA of data collected is also affected. In summary, the above results corroborate that the probing interval affects MA significantly.

Spaces discretization Since IPro is RL-based, it models its environment as a finite MDP. In order to get a finite space of states, we discretize CCO and CUC by using the results above presented (*cf.* Figs. 9 and 10) 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 ms [67]. 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 reduction of network performance [68]. 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:

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

4.2. Intelligent probing behavior

To evaluate the IPro prototype (*cf.* Algorithm 3), we test its impact (*i.e.*, the change over the time of Probing Interval) in CCO, CUC, and MA (of the throughput) in the test environment described in Section 4.1.1. Fig. 13 depicts the evaluation results, disclosing diverse facts.

Algorithm 3: Probing Interval Optimization.

Require:
Exploration parameter ε
Discount factor γ
Learning factor α
Result: A probing interval

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

- During the first 238 s (convergence time), CCO, CUC, and MA present a highly fluctuating behaviour Fig. 13(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). Fig. 13(d) illustrates how the Probing Interval changes during the learning process. As the learning process advances, the RL-agent visits each state of the space of states (*cf.* Eq. (10)) multiple times aiming at finding the most-rewarding probing strategy (*i.e.*, the convergence of the learning process).

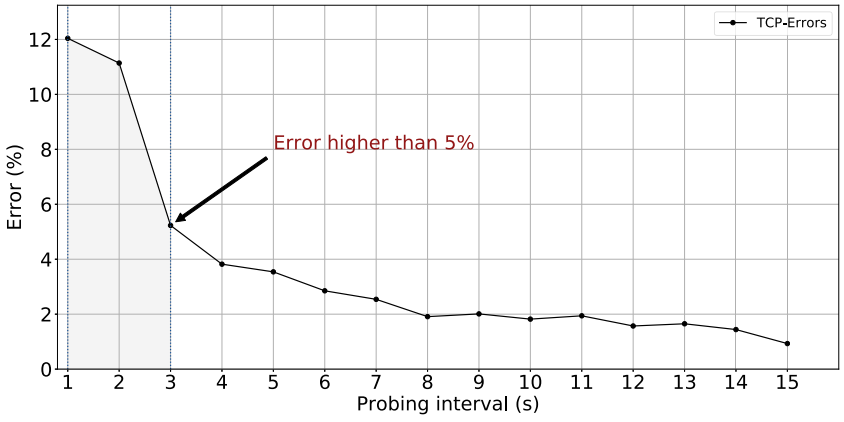


Fig. 12. TCP errors generated by Monitoring Interference.

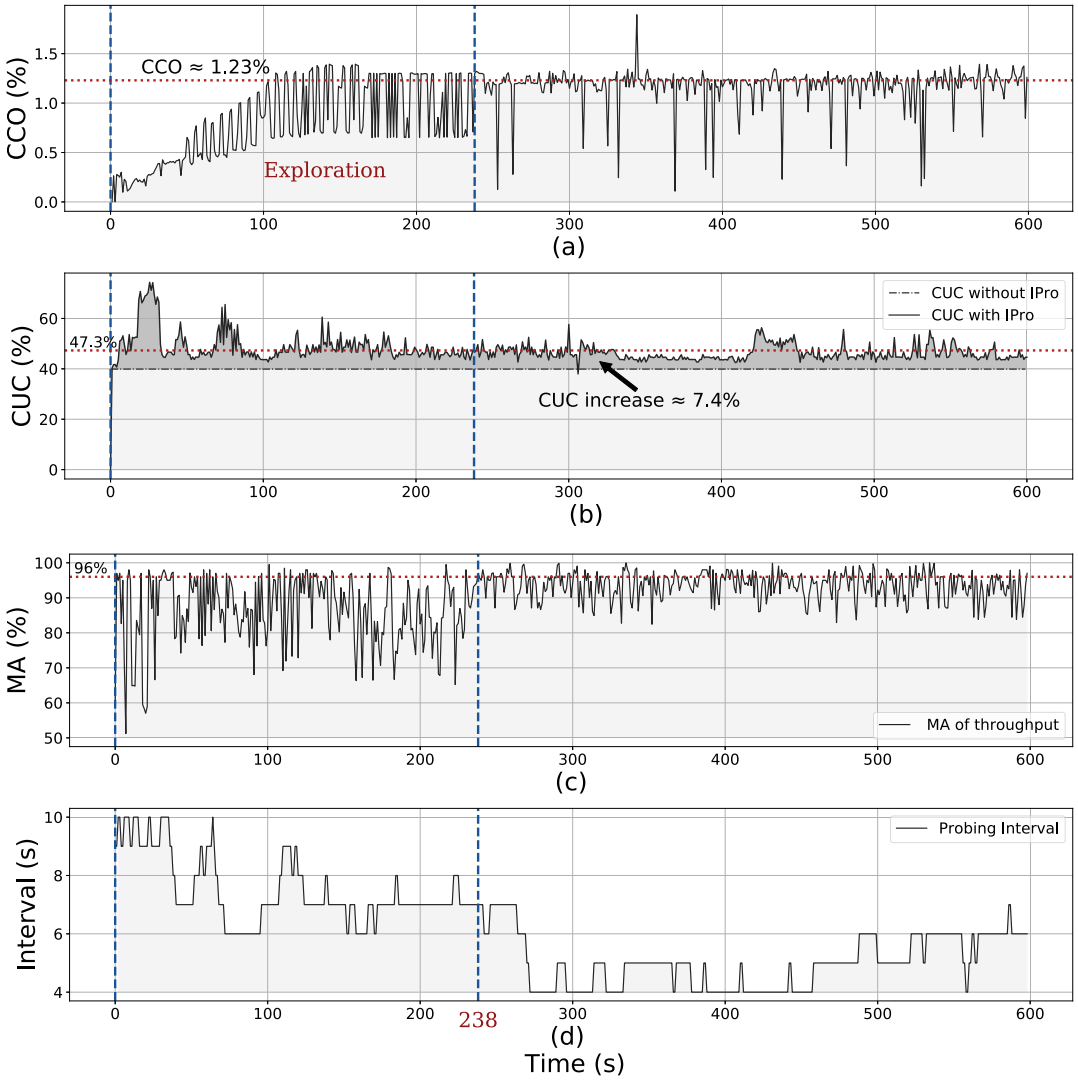


Fig. 13. Behavior of the CCO, CUC, MA, and Probing Interval.

- After the convergence time, IPro has a CCO near to 1.23%, a CUC around 7.4%, and a MA about 96%. Thus, we can state that IPro has good behavior regarding CCO, CUC, and MA. When the learning process tends to converge, the fluctuations of CCO, CUC, and MA decrease to a smaller radius. This convergence is because we choose a normal distribution function (*cf.* Eq. (4)) 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 s.
- 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, we evaluate the consumption of CPU and memory of its RL-agent. Fig. 14 depicts the evaluation results, disclosing that this RL-agent does not consume intensively the KP resources, approximately 1% -2% of CPU and 30MBytes. Therefore, we can state that the IPro RL-agent is efficient regarding CPU and memory.

4.3. Comparison

We compare IPro with 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 s (in steps of 1 s). For each interval, we performed the test 32 times during 600 s (the initial 238 s correspond to convergence time) aiming at obtaining the average of CCO, CUC, and MA. Next, we analyze only the 4, 5, and 6 s probing intervals because, in them, the MA of measurements of both throughput and delay metrics is higher than 80%. Finally, we extended the comparison of IPro with other adaptive approaches by a qualitative analysis.

Table 3
Comparison 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

4.3.1. After converging

After converging (*i.e.,* the RL-agent has learned), the experimental results (*cf.* Table 3) reveal diverse facts related to the use of RL for tuning the probing interval. First, IPro has a CCO significantly smaller than PPA. The reduction in the intervals of 4, 5, and 6 s

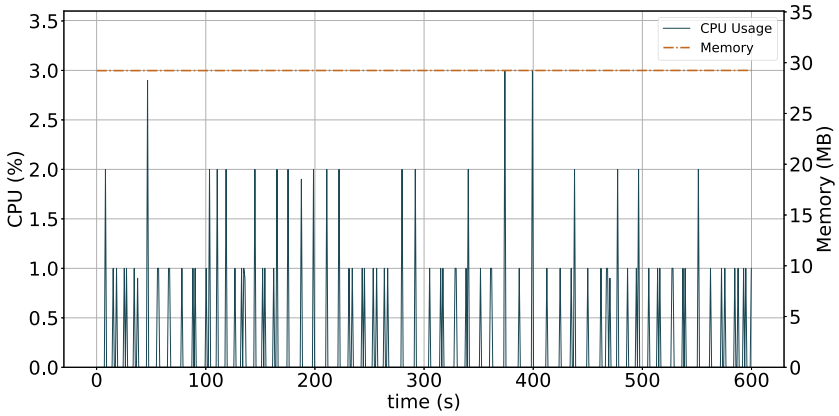


Fig. 14. Memory and CPU used by the RL-agent.

Table 4
Comparison 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

is around 16.17%, 10.22%, and 10.1%, respectively. Second, in the same intervals, IPro uses better the CPU of the controller than PPA about 13.2%, 4.3%, and 2.7%, respectively. Third, in such intervals, IPro achieves a higher MA when used to measure the throughput metric than PPA about 12.58%, 4.79%, and 9.82%, respectively. Fourth, in the analyzed intervals, 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.

4.3.2. Before converging

Before converging (*i.e.,* the RL-agent is learning), the experimental results (*cf.* Table 4) reveal diverse facts related to the use of RL for tuning the probing interval. First, 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 s, respectively. Second, 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 4.3.1, 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).

4.3.3. Qualitative analysis

Table 5 presents a qualitative comparison between IPro and other adaptive methods, disclosing diverse facts. First, methods such as [12–14] 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 accurate measurements with CCO and CPU usage negligible (*i.e.,* achieves a trade-off between MA, CCO, and CUC). Second, the methods [20,22–24] 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. Third, none of these adaptive approaches consider

Table 5
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
[12]	✓	H	H	H		Adaptive sampling algorithms are used to tune the load level generated by monitoring process
[13]	✓	H	H	H		Thresholds are used to adjust the load level generated by monitoring process
[14]	✓	H	H	H		An adaptive fetching mechanism monitors per-flow metrics, such as throughput, delay, and packet loss
[15]	✓	H	L	L		An adaptive flows statistical collection method is used to adjust the polling intervals
[20]	✓	H	L	L		A small set of matching rules and secondary controllers are used to identify and monitor aggregate flows
[22]	✓	H	L	L		A self-tuning monitoring mechanism is used to automatically adapt its settings based on the traffic dynamism
[23]	✓	H	L	H		Extra modules are included in the switches to distribute the monitoring tasks in a balanced way
[24]	✓	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

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. In this paper, we do not address this challenge. Indeed, in the IPro prototype, we use the e-greedy exploration method and do 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, we use a finite one.

5. Conclusions and future work

In this paper, we proposed an approach Called IPro that determines the probing interval that keeps CCO and CUC within acceptable theresholds. The main contributions are, first, the IPro architecture that provides a simple and efficient solution to monitor SDN-based networks by following KDN. Second, the RL-based algorithm that determines the probing interval considering network traffic variations and keeps CCO and CUC within target values. Third, the IPro prototype that achieves CCO less than 1.23%, CUC less than 8%, and an MA higher than 90% (outperforming PPA). The experimental results also indicated that IPro requires considerable time (approximately 238 s) to converge to the target state.

Future work includes exploring model-free approaches (*e.g.,* Q-learning with Experience Replay) and model-based approaches (*e.g.,* Deep Reinforcement Learning) to reduce the convergence time. We also want to correlate other parameters (*e.g.,* MA and computational resources of switches) in the reward function, aiming at improving the probing interval estimation.

Declaration of Competing Interest

We wish to draw the attention of the Editor to the following facts which may be considered as potential conflicts of interest and to significant financial contributions to this work. [OR] We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome. We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us. We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property. We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from efcastillo@unicauca.edu.co

CRediT authorship contribution statement

Edwin Ferney Castillo: Methodology, Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration. **Oscar Mauricio Caicedo Rendon:** Methodology, Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration. **Armando Ordonez:** Methodology, Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration. **Lisandro Zambenedetti Granville:** Methodology, Conceptualization, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration.

References

[1] C.C. Machado, L.Z. Granville, A. Schaeffer-Filho, 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, 2014, pp. 397–404, doi:10.1109/AINA.2014.148.

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

[3] J. d. J. Gil Herrera, J.F.B. Vega, Network functions virtualization: A survey, IEEE Latin America Transactions 14 (2) (2016) 983–997, doi:10.1109/TLA.2016.7437249.

[4] F. Estrada-Solano, A. Ordonez, L.Z. Granville, O.M.C. Rendon, A framework for sdn integrated management based on a cim model and a vertical management plane, Computer Communications 102 (2017) 150–164, doi:10.1016/j.comcom.2016.08.006.

[5] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in sdn-openflow networks, Elsevier, Computer Networks 71 (2014) 1–30, doi:10.1016/j.comnet.2014.06.002.

[6] O.M.C. Rendon, C.R.P. dos Santos, A.S. Jacobs, L.Z. Granville, Monitoring virtual nodes using mashups, Computer Networks 64 (2014) 55–70.

[7] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, FlowSense: Monitoring Network Utilization with Zero Measurement Cost, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 31–41. 10.1007/978-3-642-36516-4_4

[8] J. Suh, T.T. Kwon, C. Dixon, W. Felter, J. Carter, Opensample: A low-latency, sampling-based measurement platform for commodity sdn, in: IEEE 34th International Conference on Distributed Computing Systems, 2014, pp. 228–237, doi:10.1109/ICDCS.2014.31.

[9] M. Aslan, A. Matrawy, On the impact of network state collection on the performance of sdn applications, IEEE Communications Letters 20 (1) (2016) 5–8.

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

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

[12] S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba, Payless: A low cost network monitoring framework for software defined networks, in: IEEE Network Operations and Management Symposium, 2014, pp. 1–9, doi:10.1109/NOMS.2014.6838227.

[13] D. Raumer, L. Schwaighofer, G. Carle, Monsamp: A distributed sdn application for qos monitoring, in: Federated Conference on Computer Science and Information Systems, 2014, pp. 961–968, doi:10.15439/2014F175.

[14] N.L.M. van Adrichem, C. Doerr, F.A. Kuipers, Opennetmon: Network monitoring in openflow software-defined networks, in: IEEE Network Operations and Management Symposium, 2014, pp. 1–8, doi:10.1109/NOMS.2014.6838228.

[15] H. Tahaei, R. Salleh, S. Khan, R. Izard, K.-K.R. Choo, N.B. Anuar, A multi-objective software defined network traffic measurement, Measurement 95 (2017) 317–327.

[16] A. Tootoonchian, M. Ghobadi, Y. Ganjali, OpenTM: Traffic Matrix Estimator for OpenFlow Networks, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 201–210. 10.1007/978-3-642-12334-4_21

[17] P. Sun, M. Yu, M.J. Freedman, J. Rexford, D. Walker, Hone: Joint host-network traffic management in software-defined networks, Journal of Network and Systems Management 23 (2) (2015) 374–399, doi:10.1007/s10922-014-9321-9.

[18] X.T. Phan, K. Fukuda, Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks, Journal of Information Processing 25 (2017) 182–190.

[19] L. Liao, V.C.M. Leung, M. Chen, An efficient and accurate link latency monitoring method for low-latency software-defined networks, IEEE Transactions on Instrumentation and Measurement 68 (2) (2019) 377–391, doi:10.1109/TIM.2018.2849433.

[20] L. Jose, 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.

[21] G. Tangari, D. Tuncer, M. Charalambides, G. Pavlou, Decentralized monitoring for large-scale software-defined networks, in: IFIP/IEEE Symposium on Integrated Network and Service Management, 2017, pp. 289–297, doi:10.23919/INM.2017.7987291.

[22] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, G. Pavlou, Self-adaptive decentralized monitoring in software-defined networks, IEEE Transactions on Network and Service Management 15 (4) (2018) 1277–1291, doi:10.1109/TNSM.2018.2874813.

[23] X.T. Phan, I.D. Martinez-Casanueva, K. Fukuda, Adaptive and distributed monitoring mechanism in software-defined networks, in: 2017 13th International Conference on Network and Service Management (CNSM), 2017, pp. 1–5, doi:10.23919/CNSM.2017.8256003.

[24] H. Tahaei, R.B. Salleh, M.F. Ab Razak, K. Ko, N.B. Anuar, Cost effective network flow measurement for software defined networks: A distributed controller scenario, IEEE Access 6 (2018) 5182–5198.

[25] 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 47 (3) (2017) 2–10.

[26] P. Tsai, C. Tsai, C. Hsu, C. Yang, Network monitoring in software-defined networking: A review, IEEE Systems Journal 12 (4) (2018) 3958–3969, doi:10.1109/JSYST.2018.2798060.

[27] V. Heorhiadi, M.K. Reiter, V. Sekar, Simplifying software-defined network optimization using {SOL}, in: 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), 2016, pp. 223–237.

[28] B. Martini, D. Adami, A. Sgambelluri, M. Gharbaoui, L. Donatini, S. Giordano, P. Castoldi, An sdn orchestrator for resources chaining in cloud data centers, in: 2014 European Conference on Networks and Communications (EuCNC), IEEE, 2014, pp. 1–5.

[29] K. Benzekki, A. El Fergougui, A. Elbelrhiti Elalaoui, Software-defined networking (sdn): a survey, Security and communication networks 9 (18) (2016) 5803–5833.

[30] R. Boutaba, M.A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, O.M. Caicedo, A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, Journal of Internet Services and Applications 9 (1) (2018) 16, doi:10.1186/s13174-018-0087-2.

[31] Z.M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, K. Mizutani, State-of-the-art deep learning: Evolving machine intelligence toward tomorrow #x2019;s intelligent network traffic control systems, IEEE Communications Surveys Tutorials 19 (4) (2017) 2432–2455, doi:10.1109/COMST.2017.2707140.

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

[33] P. Wang, S.-C. Lin, M. Luo, A framework for qos-aware traffic classification using semi-supervised machine learning in sdns, in: 2016 IEEE International Conference on Services Computing (SCC), IEEE, 2016, pp. 760–765.

[34] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, H.S. Mamede, Machine learning in software defined networks: Data collection and traffic classification, in: 2016 IEEE 24th International Conference on Network Protocols (ICNP), IEEE, 2016, pp. 1–5.

[35] C. Chen-Xiao, X. Ya-Bin, Research on load balance method in sdn, International Journal of Grid and Distributed Computing 9 (1) (2016) 25–36.

[36] Y. Yu, L. Guo, Y. Liu, J. Zheng, Y. Zong, An efficient sdn-based ddos attack detection and rapid response platform in vehicular networks, IEEE Access 6 (2018) 44570–44579.

[37] D. Hu, P. Hong, Y. Chen, Fadm: Ddos flooding attack detection and mitigation system in software-defined networking, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, IEEE, 2017, pp. 1–7.

[38] T. Abar, A.B. Letaifa, S. El Asmi, Machine learning based qoe prediction in sdn networks, in: 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE, 2017, pp. 1395–1400.

[39] F. Tang, L. Li, L. Barolli, C. Tang, An efficient sampling and classification approach for flow detection in sdn-based big data centers, in: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2017, pp. 1106–1115.

[40] T.O. Ayodele, Introduction to machine learning, in: New Advances in Machine Learning, InTech, 2010.

[41] C.M. Bishop, Pattern recognition and machine learning, springer, 2006.

[42] N. Grira, M. Crucianu, 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).

[43] M. Rehman, S.A. Mehdi, Comparison of density-based clustering algorithms, Lahore College for Women University, Lahore, Pakistan, University of Management and Technology, Lahore, Pakistan (2006).

[44] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction- second edition, 1, Cambridge, Massachusetts, 2018.

[45] G. Tesauro, Temporal difference learning and td-gammon(1995).

[46] J. Yoshimoto, S. Ishii, 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), 5, IEEE, 1999, pp. 516–521.

[47] R.H. Crites, A.G. Barto, Elevator group control using multiple reinforcement learning agents, Machine learning 33 (2-3) (1998) 235–262.

[48] K. Doya, Reinforcement learning in continuous time and space, Neural computation 12 (1) (2000) 219–245.

[49] S. Schaal, C.G. Atkeson, Robot juggling: implementation of memory-based learning, IEEE Control Systems Magazine 14 (1) (1994) 57–71.

[50] A. Kolobov, Planning with markov decision processes: An ai perspective, Synthesis Lectures on Artificial Intelligence and Machine Learning 6 (1) (2012) 1–210.

[51] T.-H. Teng, A.-H. Tan, Y.-S. Tan, Self-regulating action exploration in reinforcement learning, Procedia Computer Science 13 (2012) 18–30.

[52] E. Duryea, M. Ganger, W. Hu, Exploring deep reinforcement learning with multi q-learning, Intelligent Control and Automation 7 (04) (2016) 129.

[53] S. Manju, M. Punithavalli, An analysis of q-learning algorithms with strategies of reward function.

[54] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research 4 (1996) 237–285.

[55] C.J.C.H. Watkins, Learning from delayed rewards(1989).

[56] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, 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, IEEE, 2011, pp. 1–7.

[57] A.D. Tijsma, M.M. Drugan, M.A. Wiering, Comparing exploration strategies for q-learning in random stochastic mazes, in: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1–8, doi:10.1109/SSCI.2016.7849366.

[58] N. Feamster, J. Rexford, E. Zegura, The road to sdn: An intellectual history of programmable networks, SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 87–98, doi:10.1145/2602204.2602219.

[59] S. Ayoubi, N. Limam, M. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada, O. Caicedo, Machine learning for cognitive network management, IEEE Communications Magazine (2018).

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

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

[62] C. Yu, J. Lan, Z. Guo, Y. Hu, Drom: Optimizing the routing in software-defined networks with deep reinforcement learning, IEEE Access 6 (2018) 64533–64539.

[63] J. Jiang, L. Hu, P. Hao, R. Sun, J. Hu, H. Li, Q-fdba: improving qoe fairness for video streaming, Multimedia Tools and Applications 77 (9) (2018) 10787–10806.

[64] C. Wang, L. Zhang, Z. Li, C. Jiang, Sdcor: Software defined cognitive routing for internet of vehicles, IEEE Internet of Things Journal 5 (5) (2018) 3513–3520, doi:10.1109/JIOT.2018.2812210.

[65] C.H.T. Arteaga, F. Rissoi, 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), IEEE, 2017, pp. 1–7.

[66] P. Megyesi, A. Botta, G. Aceto, A. Pescapé, S. Molnár, Challenges and solution for measuring available bandwidth in software defined networks, Elsevier, Computer Communications 99 (2017) 48–61.

[67] S. Répás, V. Horváth, G. Lencse, in: Stability analysis and performance comparison of three 6to4 relay implementations, 2015, doi:10.1109/TSP.2015.7296228.

[68] H. Xu, Z. Yu, C. Qian, X.Y. Li, Z. Liu, Minimizing flow statistics collection cost of sdn using wildcard requests, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9, doi:10.1109/INFOCOM.2017.8056992.

[69] O.N. Foundation, in: Openflow switch specification v1.3.0, ONF, 2013.

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

[71] 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, ACM, 2013, pp. 127–132.

[72] J.A. Wickboldt, W.P. De Jesus, P.H. Isolani, C.B. Both, J. Rochol, L.Z. Granville, Software-defined networking: management requirements and challenges, IEEE Communications Magazine 53(1) (2015) 278–285.

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

[74] D. Kreutz, F.M.V. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76, doi:10.1109/JPROC.2014.2371999.

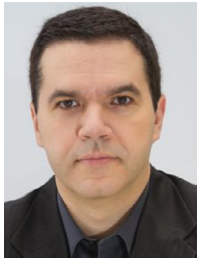
[75] P.H. Isolani, J.A. Wickboldt, C.B. Both, J. Rochol, L.Z. Granville, Interactive monitoring, visualization, and configuration of openflow-based sdn, in: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on, IEEE, 2015, pp. 207–215.

[76] L. Matignon, G. Laurent, N. Le Fort-Piat, Improving reinforcement learning speed for robot control., in: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'06., 2006, pp. 3172–3177.
[77] Z. Su, T. Wang, Y. Xia, M. Hamdi, Cemon: A cost-effective flow monitoring system in software defined networks, Computer Networks 92 (2015) 101–115.
[78] H. Tahaei, R.B. Salleh, M.F. Ab Razak, K. Ko, N.B. Anuar, Cost effective network flow measurement for software defined networks: A distributed controller scenario, IEEE Access 6 (2018) 5182–5198.
[79] B. Lantz, B. Heller, 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, ACM, 2010, p. 19.
[80] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, Internet inter-domain traffic, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 75–86.
[81] G. Maier, A. Feldmann, V. Paxson, M. Allman, On dominant characteristics of residential broadband internet traffic, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, ACM, 2009, pp. 90–102.
[82] C. Müller, C. Timmerer, A vlc media player plugin enabling dynamic adaptive streaming over http, in: Proceedings of the 19th ACM International Conference on Multimedia, in: MM '11, ACM, New York, NY, USA, 2011, pp. 723–726, doi:10.1145/2072298.2072429.
[83] A. Mockus, R.T. Fielding, J. Herbsleb, A case study of open source software development: The apache server, in: Proceedings of the 22Nd International Conference on Software Engineering, in: ICSE '00, ACM, New York, NY, USA, 2000, pp. 263–272, doi:10.1145/337180.337209.
[84] E.F. Castillo, intelligentProbing prototype and results, Accessed November 09, 2019.
[85] B.A.A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Communications Surveys & Tutorials 16(3) 1617–1634.

Edwin Ferney Castillo is an Electronic and Telecommunications Engineer from the University of Cauca. He is currently a Master student in Telematics Engineering at University of Cauca. He is a member of the Telematics Engineering Group of University of Cauca. His topics of interest include network and service management, web technologies, SDN, NFV, IoT, and Machine Learning for Networking.

Oscar Mauricio Caicedo Rendon is a full professor at the Telematics Department of UNICAUCA. He is a member of the Telematics Engineering Group of UNICAUCA, and the Computer Networks Group of the Federal University of Rio Grande do Sul (UFRGS). He hold a Ph.D. in Computer Science from the Institute of Informatics of UFRGS, Brazil, 2015, a Master in Telematics and a Bachelor in Telecommunications from UNICAUCA in 2006 and 2001, respectively. He has been an IETF Fellowship (2016) and a traveler grant from ACM Sigcomm (2016). Furthermore, he has published in prominent journals as IEEE Communications Magazine, Elsevier Computer Networks and Elsevier Computer Communications, and in relevant conferences as IEEE Globecom, CNSM, AINA, COMPSAC, and ISCC. His topics of interest include network and service management, Web 2.0/3.0 technologies, SDN, Data Centers, NFV, and Machine Learning for Networking.

Armando Ordonez is a Ph.D. in Telematics Engineering at the University of Cauca, Colombia. He is currently a postdoctoral fellow at University of Cauca. He received his M.Sc. degree in Telematics Engineering (2009) and his Bachelor's degree in Electronics and Telecommunications Engineering (2003) from the University of Cauca. His topics of interest include software-defined networking, and machine learning.



Lisandro Zambenedetti Granville is an Associate Professor of Computer Science at the Institute of Informatics of the Federal University of Rio Grande do Sul (UFRGS), Brazil. He holds a Ph.D. (2001) and an M.Sc. (1998) degrees in Computer Science, both received from UFRGS. From September 2007 to August 2008 he was a visiting researcher at the University of Twente, The Netherlands, with the Design and Analysis of Communication Systems group. He is a member of the Computer Networks Group, where I develop research projects on network and service management. As an Associate Professor, he is also involved with supervision and education activities on undergraduate and graduate courses in both Computer Science and Computer Engineering.