Detection-Aware Controller Placement in Software-Defined Networks

Loïc Desgeorges, Francesco Bronzino
ENS de Lyon, CNRS, Univ. Claude Bernard Lyon 1, LIP, UMR 5668
Lyon, France.
loic.desgeorges@univ-lyon1.fr, francesco.bronzino@ens-lyon.fr

Francescomaria Faticanti
Univ. Lille, Inria, CNRS UMR 9189 CRIStAL
Lille, France.
francescomaria.faticanti@inria.fr

***Abstract*—Software-Defined Networking (SDN) decouples control and data planes, making controller orchestration a central challenge known as the Controller Placement Problem (CPP). While security-aware CPP has been studied mainly through active mechanisms (e.g., consensus), their assumptions do not hold for passive mechanisms such as anomaly detection. Yet, security tasks increase controller load, and prior analytical and experimental works show that control-plane contention degrades network performance. This paper investigates the impact of passive security—specifically anomaly detection—on controller load and integrates the choice of detection model into the CPP to jointly optimize security and performance. Experiments on the Ryu controller over the GEANT topology show that passive mechanisms can reduce effective controller capacity by up to a half. Compared to classical strategies, our approach improves security by 11% with only 3% performance overhead, whereas the most secure baseline incurs over 67% performance degradation for a 16% security gain.**

***Index Terms*—Controller Placement Problem, Security, Detection, Software-Defined Networking**

I. Introduction

Software-Defined Networking (SDN) [1] enables programmable network architectures by decoupling the data plane from the control plane and centralizing the latter through software-based controllers. This paradigm simplifies network management, improves energy efficiency, and supports advanced applications [1]. SDN is now a key component of modern 5G and emerging 6G networks, where it supports network slicing, edge computing, and heterogeneous access technologies [2].

Early SDN deployments relied on a single centralized controller, which quickly exposed limitations in scalability, reliability, and fault tolerance [1]. Distributed controller architectures were therefore introduced, combining logical centralization with physical distribution. In such architectures, determining how many controllers to deploy and where to place them becomes a critical design problem, known as the Controller Placement Problem (CPP) [3]. Most CPP formulations focus on performance-related objectives such as latency minimization [3], reliability through redundancy [4], load balancing, energy efficiency [5], or multi-objective trade-offs involving cost and resilience. Despite extensive research, the impact of security mechanisms on controller placement remains largely unexplored, although recent surveys highlight its growing importance [6], [7].

Securing the SDN control plane further complicates its design, as controllers are high-value attack targets and often rely on anomaly-detection mechanisms [1], [8], [9]. In practice, security is typically added after controller architectures are optimized, implicitly assuming that controller resources remain fully available for control tasks. However, security tasks introduce additional overhead on the controller, which adds to an already fully utilized system [10], and queueing-theoretic analyses as well as experimental studies have shown that contention among control-plane requests leads to overload, increased processing delays, and degraded network performance [11], [12].

As we show later, enabling anomaly detection at fine time granularities can significantly reduce the effective processing capacity of SDN controllers —by more than half in our experiments— thereby challenging existing controller placement and provisioning (CPP) models. More broadly, anomaly-detection algorithms, often based on machine learning, introduce non-negligible computational overhead and compete with core control-plane functions in large-scale WAN, 5G/6G, and IoT environments [10], [13]. Despite this, the impact of detection mechanisms on controller load and effective capacity is rarely considered in CPP formulations, potentially leading to inefficient or unstable deployments.

In this paper, we address this gap by explicitly modeling the computational overhead of anomaly-detection mechanisms and integrating it into the controller placement process. Rather than proposing a new detection algorithm, we quantify and measure how different detection models and execution strategies affect controller capacity and incorporate this impact into an extended CPP formulation. We introduce the Controller and Detection Placement Problem (CDPP), which jointly determines controller placement, controller type, and detection model selection to balance control-plane performance, deployment cost, and security level.

Using experimental measurements on a real SDN controller and evaluations on realistic network topologies, we show that ignoring the cost of passive security mechanisms can significantly degrade control-plane performance, whereas detection-aware placement enables substantial security gains with only marginal performance overhead.

The remainder of this paper is organized as follows. Section II introduces detection-aware controller placement,

Section III presents the CDPP formulation, and Section IV evaluates the proposed approach.

II. Introducing the detection in the Controller Placement Problem

A. Description

The partitioning of the SDN control plane raises challenges related to latency, reliability, and load balancing, which are central to the CPP. CPP determines the number and location of controllers and the assignment of switches to controllers. These decisions directly impact control-plane latency, scalability, and overall network performance. Similar issues arise in SDN-based 5G/6G networks, where virtualized control functions run on general-purpose CPUs under tight resource constraints.

The SDN control plane constitutes a high-value attack surface, with vulnerabilities at both controller level and East–West interfaces [1]. To mitigate these threats, anomaly-detection mechanisms are often embedded within controllers.

However, SDN controllers already operate on heavily loaded general-purpose CPUs. Prior studies report near-100% CPU utilization under realistic workloads [10], and several dedicated cores may be required to maintain acceptable performance [14]. Since core control-plane functions such as flow-rule installation, topology discovery, and path computation are computationally expensive, even limited additional processing can increase latency and degrade responsiveness.

In this context, deploying neural-network-based anomaly detection further reduces available controller capacity and may destabilize the control plane. Neural networks nevertheless remain widely adopted for SDN anomaly detection [8], [9], creating a tension between security requirements and limited controller resources [13].

This paper therefore studies how integrating anomaly-detection mechanisms affects controller placement decisions. In particular, we investigate how detection-induced processing overhead reduces controller capacity and how this reduction should be incorporated into CPP formulations

B. Detection strategy

There are several ways to orchestrate anomaly detection at the control plane. Models range from statistical and rule-based approaches to Machine Learning techniques, with Neural Networks becoming particularly popular in SDN controllers [8], [9], [15]. These models differ in computational cost, memory footprint, input requirements, and inference latency, directly affecting the overhead imposed on the control plane.

Another key factor is the detection pipeline, i.e., when and how frequently the anomaly detection algorithm processes incoming control traffic. The same model can have very different performance impacts depending on this orchestration strategy. A first approach is to execute the detection algorithm per packet, as in [8], but such an approach does not scale to large-scale networks and is typically limited to highly sensitive contexts requiring fine-grained traffic inspection.

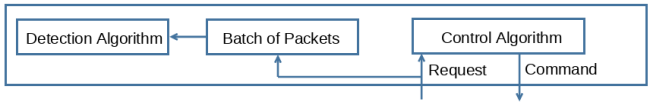

Fig. 1: Control pipeline in case of a detection algorithm apply periodically.

A second approach is to perform anomaly detection periodically on batches of packets as in [9], [15] and as illustrated in Fig. 1. Instead of analyzing each packet individually, the controller collects a set of packets during a predefined time window and runs the detection algorithm at fixed intervals. This strategy reduces the frequency of model executions, thereby lowering the resource consumption on the controller. However, it increases the detection delay: anomalies occurring between two detection periods may remain undetected for longer. Furthermore, batching packets increases the volume of input data provided to the model at once, potentially increasing inference time even if the number of executions decreases.

As highlighted in [16], configuring anomaly-detection models involves a trade-off between inference time and detection accuracy. Hence, increased accuracy usually requires additional processing, which introduces latency and reduces controller capacity. In our measurements, periodic detection with a 10 ms period reduces capacity by more than half (Table III).

Therefore, detection mechanisms must be treated as first-class components of control-plane design. Model choice and execution strategy should be considered alongside controller placement. This motivates the Controller and Detection Placement Problem (CDPP), extending classical CPP by explicitly modeling detection overhead and jointly optimizing controller placement, type, and detection model under performance, cost, and security constraints.

III. Formalisation of the CDPP

Determining the optimisation placement of the controllers is a common problem from the literature but we propose to introduce the choice of the detection model in the Controller and Detection Placement Problem (CDPP).

System Model. We have an SDN network topology consisting of a set of nodes $\mathcal{N}$. Each node of the network can contain a switch or a controller. The set of switches $S \subset N$ is initially known, while the set of controllers $\mathcal{C} \subseteq \mathcal{N}$ has to be determined. Each controller can be instantiated by choosing one of $\mathcal{K}$ different types, where each type of controller k is characterized by its computational capability PP_k and its price c_k. The set of models that can be implemented in the controller is denoted $\mathcal{M}$ and is detailed in Table I of section IV. We assumed that each model $m \in \mathcal{M}$ has a F1-score F_m.

Variables. We define three kinds of binary variables.

1) Controller placement. We define $y_{c,k} \in \{0,1\}$, $\forall c \in \mathcal{N}, \forall k \in \mathcal{K}$. This variable is equal to 1 if a controller of type k is placed on node c.
2) Switch-controller connection. The variable $z_{s,c} \in \{0,1\}$, $\forall c \in \mathcal{N}, \forall s \in \mathcal{S}$, represents the physical connection between a switch and a controller. It is equal to one if controller c communicates with switch s.
3) Detection model. The variable $d_{c,m} \in \{0,1\}$, $\forall c \in \mathcal{N}, \forall m \in \mathcal{M}$, represents the detection model m implemented in the controller c. It is equal to one if controller c runs the detection model m with F1-score F_m.
Objective. The objective is twofold: i) minimize the latency over the network, and ii) minimize the cost of the control plane, to limit the number of controllers used. Hence, the objective function is the sum of these normalized components defined as follows:

$$\sum_{c\in\mathcal{N}}(\sum_{s\in\mathcal{S}}\hat{l}_{c,s}+\sum_{c'\in\mathcal{N}}\hat{l}_{c,c'})+\sum_{c\in\mathcal{N}}\sum_{k\in\mathcal{K}}\hat{ct}_{k,c}. \quad (1)$$

In equation 1, $\hat{l}_{c,s}$, $\hat{l}_{c,c'}$ and $\hat{ct}_{k,c}$, represent the normalized values of latency between switches and controllers, latency between controllers, and deployment cost of controllers, respectively. Normalization ensures that the different components are expressed on comparable scales; in our evaluation, each term is normalized by its maximum observed value in the network. Each component of (1) is defined in the following.
Latency between switches and controllers. This latency corresponds to the processing time of the switches' requests to determine the commands in response. This time, namely $l_{c,s}$, consists of the physical latency and the computation time.

$$l_{c,s} = l_{phy,c,s} + l_{comp,c,s} \quad (2)$$

The physical latency, $l_{phy,c,s}$, corresponds to the transmission time on the link between switch s and controller c. We assume that the switch communicates with only one controller and $l_{phy,c,s} = \sum_{n\in\mathcal{N}} z_{s,n} \times l_{phy,n,s}$. The computation time, $l_{comp,c,s}$ depends on the application considered and the security mechanism implemented. For instance in case of a detection algorithm applied at each packet then the computation time increases but we will not consider such applications in this paper.
Latency between controllers. Further, we need to consider the latency between different controllers. Indeed, to ensure consistency in control and provide a global view to all controllers, they must exchange information related to the subdomains they are responsible for certain tasks, and the latency $l_{c,c'}$ between two controllers depends on the nodes (n and n') on which they are located:

$$l_{c,c'} = (\sum_{n\in\mathcal{N}}\sum_{n'\in\mathcal{N}} l_{n,n'} \times y_{c,n}y_{c',n'}), \quad \forall (c,c') \in \mathcal{N}^2 \quad (3)$$

Cost. The model minimizes control-plane cost by optimizing controller placement and selection. We consider four controller types with distinct capacities, following the classification in [3] (Table III).

However, the effective processing capacity also depends on other applications running at the control plane, particularly security mechanisms. For instance, when a detection algorithm is activated it consumes additional resources (e.g., CPU cycles), thereby reducing the capacity available for network control tasks. Consequently, the effective processing power of a controller decreases when a detection model is executed. The detailed computation of this impact is provided in section IV.

Finally, the second component of the objective function presented in (1) is:

$$ct_{k,c} = cost_k \cdot y_{c,k}, \quad (4)$$

where $cost_k$ is the price of a single controller of type k.
Model constraints. The following set of constraints over the control plane needs to be considered:
Each node can host at maximum 1 controller:

$$\sum_{k\in\mathcal{K}} y_{c,k} \leq 1, \quad \forall c \in \mathcal{N}. \quad (5)$$

Each switch can be connected only to a node where a controller is placed:

$$z_{s,c} \leq \sum_{k\in\mathcal{K}} y_{c,k}, \quad \forall c \in \mathcal{N}, \forall s \in \mathcal{S}. \quad (6)$$

Each switch is connected to exactly one controller:

$$\sum_{c\in\mathcal{N}} z_{s,c} = 1, \quad \forall s \in \mathcal{S}. \quad (7)$$

We assumed that each controller has one detection model implemented:

$$\sum_{m\in\mathcal{M}} d_{c,m} = \sum_{k\in\mathcal{K}} y_{c,k}, \quad \forall c \in \mathcal{N}. \quad (8)$$

Then, we introduced α the security level. Here, to ease the readability we assume that each controller has the same level of criticality and so the security level corresponds to the minimum average F1-score of the detection model implemented on the controllers. It can also be a weighting if some controllers are more critical. For instance, if $\alpha = 0$, no detection mechanism is applied. Here, we set the constraint as the average detection F1-score of the models on the controllers is at least α:

$$\sum_{m\in\mathcal{M}} d_{c,m} \times F_m \geq \alpha \cdot \sum_{k\in\mathcal{K}} y_{c,k}, \quad \forall c \in \mathcal{N}. \quad (9)$$

It should be noted that, the detection delay affects the application response time itself (especially in case of a per packet detection) which is a concern in case of delay-sensitive applications,. Therefore, it is possible to add a constraint on the computation time of the detection algorithm's model.

The implementation of a detection model affects the capacity of the controller and so the choice of the category of the controller level. The category of the controller defined the capacity of the controller and so the space left for the detection algorithm. The total rate received from switches

cannot exceed the total processing capacity of a given controller:

$$\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{M}} y_{c,k}\cdot d_{c,m}\cdot PP_{k,m} \geq \sum_{s\in\mathcal{S}} z_{s,c}\cdot PR_{sw}, \quad \forall c\in\mathcal{N}, \tag{10}$$

where PR_{sw} refers to the packet rate of each switch (assuming all switches have the same packet rate) and $PP_{k,m}$ refers to the processing rate of the controller of category k with the concurrence of the detection model m. Formally, the processing latency $l_{comp,c,s}$ is inversely related to the processing power $PP_{k,m}$ of the controller running model m; higher $l_{\text{comp},c,s}$ implies lower $PP_{k,m}$.

Classical CPP formulations assume that controllers have fixed processing capacity and ignore the overhead of anomaly detection. In contrast, with CDPP we explicitly model the cost and impact of detection mechanisms in the control plane and jointly optimize controller and detection placement.

IV. Results

In this section we study the impact of the detection model on the control-plane performance. First, we discuss the trade-off between model accuracy and computation time. Next, we analyze its effect on controller response time. Finally, we examine the implication on the network performance and security based on the control-plane deployment.

A. Experimental Setup

We run experiments using the Ryu controller with a routing strategy [17] and traffic from the Totem dataset [18] over the GÉANT 2012 topology (40 nodes, 61 links) [19]. Simulated attacks included switch takeovers, DDoS, unauthorized rule changes, and corrupted rules. The dataset's first half represents nominal traffic, and the second half contains randomly launched attacks.

As a reminder, the goal of this work is not to design a new detection algorithm but to quantify and integrate the computational cost of detection mechanisms into the CDPP. Since Recurrent Neural Networks (RNNs) are widely used for SDN anomaly detection [8], [9], we focus on this family of models. Five RNN-based detection models are evaluated (Table I), and the trade-off between detection accuracy and computation time is analyzed, as these metrics jointly determine the effectiveness of the detection mechanism and the overhead imposed on the SDN controller. In our experiments, the controller is limited to at most eight CPU cores, following the architectural constraints reported in [14]. The resulting Pareto front highlights the trade-off between detection accuracy and computational overhead. Similar experiments with Convolutional Neural Networks yield comparable trends; due to space constraints, only RNN results are reported.

Computation time drives the reactivity of the detection algorithm: longer processing means slower detection and more time for attackers to act. In Table I, model 6 offers the highest F1-score but the lowest reactivity, while the fastest model has the poorest F1-score. Choosing a model therefore requires balancing F1-score and responsiveness, as illustrated by model 3 or 4. These different choices impact the controller performance, since more complex models use more CPU and can degrade control-plane operations.

TABLE I: Number of neurons per layer of the detection models for the RNN configuration and their performance in terms of F1-score and time computation of the model.

Model	L1 Simple	L2 GRU	L3 LSTM	L4 Dense	F1-s (%)	Time (ms)
1	2	2	1	1	46	8.56
2	16	16	8	8	78	9.23
3	128	128	64	64	87	9.51
4	256	256	128	96	90	11.02
5	512	512	256	128	92	15.38
6	1024	512	256	128	93	16.52

B. Impact on the controller

The use of such models have a direct impact on the capacity of the controller depending on the frequency of the computation. We analyze the impact of periodic anomaly detection on controller performance and capacity. This detection strategy, which is widely adopted in the literature due to its scalability, consists in executing the detection algorithm at fixed intervals of P seconds through the set of packets collected by the controller. To evaluate its impact on controller performance, we deployed the proposed models on the Ryu controller and measured the average controller response time as a function of the detection model and the number of CPU cores allocated to the controller. The results, with its confidence interval at 95%, are shown in Fig. 2 and in these experiments, we considered a detection period of $P = 100ms$.

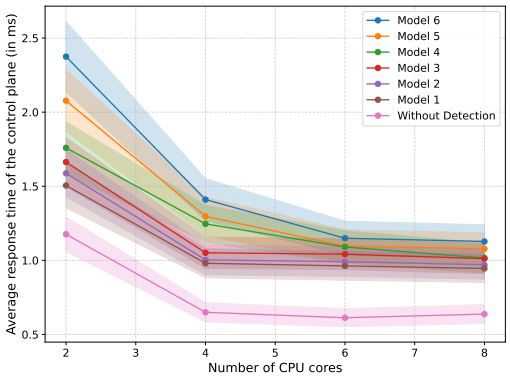

Fig. 2: Average response time of the controller depending on the anomaly detection model used.

Fig. 2 represents the average response time of the controller depending on the detection model used. For example, the blue curves is associated to Model 6 defined in Table. I as the heaviest model (the one on the last line). This figure shows that controller response time decreases with the number of CPU cores, but the marginal gain becomes

negligible beyond six cores. We therefore use eight cores in the following experiments to remain in the saturation zone while avoiding unnecessary resource over-provisioning. Although a 1 ms increase in the control plane response time may appear negligible in isolation, queueing theory [11] and experimental evaluations [12] show that such an increase can double controller utilization and push the control plane into saturation. Also, it shows that enabling the detection algorithm significantly affects the controller's response time, resulting in higher latency on the network. This increase can be attributed to resource sharing and heightened CPU contention introduced by the detection process. While allocating additional CPU cores mitigates the performance degradation by reducing the gap between response times with and without detection, the overhead induced by the detection algorithm remains noticeable.

To quantify the impact of the parameter P of the detection mechanism on controller load we measure the average response time of the controller depending on P. The results are shown in Table. II. The choice of the period has been made according to the ones of [20].

TABLE II: Average controller response time (ms) depending on the detection period P, measured with 8 CPU cores.

Model	**Detection period** P		
	10	100	1000
Model 6	1.84	1.13	1.02
Model 5	1.74	1.08	1.01
Model 4	1.69	1.02	0.98
Model 3	1.61	1.01	0.89
Model 2	1.60	0.97	0.82
Model 1	1.60	0.94	0.81

The detection period P governs the trade-off between execution frequency and per-execution workload. Small values of P increase the number of detections per unit time and lead to CPU contention, whereas large values of P increase batch size and per-execution cost. As shown in Table II, the controller response time decreases as the period P increases. Although the detection time increases with P, the detection process is executed in parallel with the control computation, as illustrated in Fig. 1. Consequently, the observed variation in controller response time with P is mainly due to resource contention with the detection algorithm. As P increases, contention events become less frequent, but their duration increases because of the larger batch size.

Although absolute values are close, relative differences are significant, as doubling the per-packet processing time halves controller capacity. Since controller capacity is defined as $PP = 1/PT$, the additional processing introduced by anomaly detection—especially with shorter detection periods—increases PT (Table II) and reduces capacity.

Based on these measures, the result of the processing power (i.e., capacity) of the controller depening on the detection model apply is given in Table. III.

It can be observed from Table III that the controller capacity is influenced by both the detection model and the selected detection strategy. These results illustrate the trade-off between operational cost and control-plane security, which directly affects the controller deployment strategy. For example, considering Model 3, which provides a favorable balance between F1-score and computation time, a detection period of $P = 10ms$ divides the controller's capacity by a factor greater than 2.6, whereas for $P = 1000ms$ the capacity is divided by a factor of approximately 1.5.

A trade-off is also observed between detection reactivity and controller performance. Increasing the reactivity of the detection mechanism requires reducing the detection period, which in turn leads to higher controller load and so decreases its capacity.

C. Performance of the controllers deployment

Such a modification of the controller capacity directly influences the performance of the control plane. To prove it, we compare the deployment of a SDN control plane with three strategies:

- *w/o detection - strategy 1*: the detection mechanism is not considered in the CPP; instead, once the load on each controller is known, the appropriate model is selected afterward.
- *w/o detection - strategy 2*: the detection mechanism is not considered in the CPP; however, once the control plane is determined, the model with the highest F1-score that can still be supported is assigned to each controller.
- *CDPP*: the detection is taken into account in the constraint of the CPP (i.e., it corresponds to the model described in section II and the table used to determine the capacity of the controller is the one of Table. III).

For the first two strategies, we solve the CPP without constraint 8 and (9) and without the decision variable d. We assume that the period is fixed at $P = 100$ ms. We used the Gurobi solver computation [21].

This evaluation is performed on the 83 topologies from the Zoo dataset that contain more than 10 nodes [19]. This real-world wide-area network provides a representative testbed for evaluating the scalability and efficiency of controller placement strategies.

The results—namely, the performance of the control plane (value of the objective function, eq. 1) and the average F1-score of the deployed model—are reported in Table IV. The margin of the 95% confidence interval is given. The two last columns (*strategy two* and *detection*) provide the relative gap with respect to *strategy one*.

TABLE IV: Relative gap of control-plane performance and security compared to *Strategy one*.

Metric	**w/o detection**		*CDPP*
	Strategy 1	*Strategy 2*	
Performance (Obj Function)	33 ($\pm$0.3)	50 ($\pm$0.3) *(+67%)*	34 ($\pm$0.3) ($+\mathbf{3}\%$)
Detection F1-score	81% ($\pm$4)	92% ($\pm$0.04) *(+13%)*	90% ($\pm$1.2) ($+\mathbf{11}\%$)

TABLE III: Performance characteristics of network devices with detection algorithms. Processing power shown in Kpps (kilo packets per second). For models with detection, values correspond to detection periods P (algorithm applied every Ps). Note that subcolumns headers (10ms, 100ms, 1000ms) represent detection period P values.

Device		**Baseline**	**Model 1**			**Model 2**			**Model 3**			**Model 4**			**Model 5**			**Model 6**		
Type	**Price**	**No Det.**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**	**10**	**100**	**1000**
Switch	—	2		—			—			—			—			—			—	
Ctrl. Type 1	\$1.2K	25	10	16	19	10	16	18	9	15	17	8	14	15	8	13	15	8	13	14
Ctrl. Type 2	\$2.5K	40	15	26	30	15	25	29	15	24	27	14	24	25	14	22	23	13	21	23
Ctrl. Type 3	\$6.5K	80	30	51	60	30	49	58	29	47	54	28	47	49	27	44	47	26	42	47
Ctrl. Type 4	\$12K	150	56	96	111	56	92	108	55	89	101	53	88	92	51	83	89	49	79	88

Since *strategy 1* solves the problem with the fewest constraints, it achieves the best final objective, i.e., lowest cost and latency, but at the expense of a suboptimal detection model. *Strategy 2* optimizes the detection F1-score in two steps but severely degrades control-plane performance, raising the objective by over 67%. Our proposed strategy strikes a better balance: a small objective increase (3%) yields an 11% F1-score gain.

Thus, integrating detection from the outset provides the best trade-off between security and control-plane performance, while ignoring it leads to either degraded performance or insufficient security. Security should therefore be incorporated into the CPP, even for passive mechanisms such as detection algorithms.

V. Conclusion

This paper studies the impact of integrating detection strategies into the SDN control plane, analyzing the trade-off between accuracy and computation, quantifying costs, and their effect on controller deployment. Results show that even passive security mechanisms improve security with minimal control-plane impact. The proposed methodology is general and can be applied to various SDN-based networks, including emerging 5G and future 6G architectures, by adapting system parameters to network-specific requirements.

Future work includes evaluating additional detection models, such as Support Vector Machines and Decision Trees, extending the formulation to jointly consider both passive (e.g., anomaly detection) and active (e.g., replication or consensus) security mechanisms, and addressing practical deployment challenges, including variable processing capacities, network latency fluctuations, and controller reliability, whose impact on the control plane differs.

Acknowledgment

This work was funded by the Université Claude Bernard under the project AAP Accueil EC.

References

[1] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 2014.

[2] Alcardo Alex Barakabitze and Ray Walshe. Sdn and nfv for qoe-driven multimedia services delivery: The road towards 6g and beyond networks. *Computer Networks*, 214:109133, 2022.

[3] Abdullah Naseri, Mahmood Ahmadi, and Latif PourKarimi. Placement of sdn controllers based on network setup cost and latency of control packets. *Computer Communications*, 208:15–28, 2023.

[4] Abeer AZ Ibrahim, Fazirulhisyam Hashim, Aduwati Sali, Nor K Noordin, Keivan Navaie, and Saber ME Fadul. Reliability-aware swarm based multi-objective optimization for controller placement in distributed sdn architecture. *Digital Communications and Networks*, 2024.

[5] Abeer AZ Ibrahim, Fazirulhisyam Hashim, Nor K Noordin, Aduwati Sali, Keivan Navaie, and Saber ME Fadul. Heuristic resource allocation algorithm for controller placement in multi-control 5g based on sdn/nfv architecture. *IEEE Access*, 2020.

[6] Abdulrahman M Abdulghani, Azizol Abdullah, AR Rahiman, Nor Asilah Wati Abdul Hamid, Bilal Omar Akram, and Hafsa Raissouli. Navigating the complexities of controller placement in sd-wans: A multi-objective perspective on current trends and future challenges. *Computer Systems Science & Engineering*, 49, 2025.

[7] Tasneem Darwish, Taqwa Ahmed Alhaj, and Fatin A Elhaj. Controller placement in software defined emerging networks: a review and future directions. *Telecommunication Systems*, 88(1):18, 2025.

[8] Loïc Desgeorges, Jean-Philippe Georges, and Thierry Divoux. Detection of anomalies of a non-deterministic software-defined networking control. *Computers & Security*, 129:103228, 2023.

[9] Yang Xu and Yong Liu. Ddos attack detection under sdn context. In *IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications*, pages 1–9. IEEE, 2016.

[10] Burak Görkemli, Sinan Tatlıcıoğlu, A Murat Tekalp, Seyhan Civanlar, and Erhan Lokman. Dynamic control plane for sdn at scale. *IEEE Journal on Selected Areas in Communications*, 36, 2018.

[11] Bing Xiong, Kun Yang, Jinyuan Zhao, Wei Li, and Keqin Li. Performance evaluation of openflow-based software-defined networks based on queueing model. *Computer Networks*, 102:172–185, 2016.

[12] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[13] Qinwen Hu, Se-Young Yu, and Muhammad Rizwan Asghar. Analysing performance issues of open-source intrusion detection systems in high-speed networks. *Journal of Information Security and Applications*, 2020.

[14] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, et al. Orion: Google's {Software-Defined} networking control plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 83–98, 2021.

[15] Tohid Jafarian, Ali Ghaffari, Ali Seyfollahi, and Bahman Arasteh. Detecting and mitigating security anomalies in software-defined networking (sdn) using gradient-boosted trees and floodlight controller characteristics. *Computer Standards & Interfaces*, 91:103871, 2025.

[16] Gerry Wan, Shinan Liu, Francesco Bronzino, Nick Feamster, and Zakir Durumeric. Cato:end-to-end optimization of ml-based traffic analysis pipelines. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025.

[17] Ryu Project Team. *https://www.opennetworking.org/ , last visit 13/01/2026*, 2012.

[18] TOTEM. *https://totem.info.ucl.ac.be/index.html , last visit 13/01/2026*, January, 2006.

[19] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[20] Mohammad A Salahuddin, Vahid Pourahmadi, Hyame Assem Alameddine, Md Faizul Bari, and Raouf Boutaba. Chronos: Ddos attack detection using time-based autoencoder. *IEEE Transactions on Network and Service Management*, 19(1):627–641, 2021.

[21] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2025.