

No binding machine learning architecture for SDN controllers

Wael Hosny Fouad Aly, Hassan Kanj, Nour Mostafa, Zakwan Al-Arnaout, Hassan Harb

College of Engineering and Technology, American University of the Middle East, Egaila, Kuwait

Article Info

Article history:

Received Mar 19, 2024

Revised Nov 4, 2024

Accepted Mar 9, 2025

Keywords:

Artificial intelligence

Controller placement

Machine learning

Neural networks

Software-defined networking

ABSTRACT

Although software-defined networking (SDN) has improved the network management process, but challenges persist in achieving efficient load balancing among distributed controllers. Present architectures often suffer from uneven load distribution, leading to significant performance deterioration. While dynamic binding mechanisms have been explored to address this issue, these mechanisms are complex and introduce a significant latency. This paper proposes $SDN_{CTRL_{ML}}$, a novel approach that applies machine learning mechanisms to improve load balancing. $SDN_{CTRL_{ML}}$ introduces a scheduling layer that dynamically assigns flow requests to controllers using machine learning scheduling algorithms. Unlike previous approaches, $SDN_{CTRL_{ML}}$ integrates with the standard SDN switches and adapts to different scheduling algorithms, minimizing disruption and network delays. Experimental results show that $SDN_{CTRL_{ML}}$ has outperformed static-binding controllers models without adding complexities of dynamic-binding systems.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Wael Hosny Fouad Aly

College of Engineering and Technology, American University of the Middle East

Egaila 54200, Kuwait

Email: wael.aly@aum.edu.kw

1. INTRODUCTION

Scalable network architectures are important in various network applications. They enable the provision of reliable and sufficient services for specific types of traffic [1]. Scalability could be achieved by maintaining an overview of the states and conditions of networks worldwide as observed from a broader perspective by regulating the flow of the network traffic across underlying layers [2]. This has led to significant changes in network design and management [3]. A prominent example of such a deployment is the Ethane project [4], which introduces a network paradigm for SDNs that utilizes a central controller managing policy at the flow level. According to Almadani *et al.* [5], the SDN paradigm has the advantage of separating the control plane from the data plane, which leads to a faster and more dynamic approach compared to traditional network architectures [6]. Prabakaran *et al.* [7] suggest that the control plane could be split into multiple virtual networks implementing different policies. This approach allows the SDN paradigm to address networking issues from different perspectives [8], and to meet the requirements of emerging technologies like IoT and 5G [9]. The adoption of the SDN paradigm depends on its success in providing solutions to problems that could not be addressed through conventional networking protocols and architectures. Large companies such as Microsoft and Google have already implemented the SDN paradigm in their data centers [10]-[13].

The SDN architecture is composed of three main planes: the *data*, *control*, and *application* planes,

as depicted in Figure 1. The data plane is responsible for forwarding packets, the control plane decides how packets should be forwarded, and the application plane hosts network services [14]. The northbound API facilitates communication between the control and application planes, enabling developers to build applications without needing detailed knowledge of the controller operations within the data plane [15]. Different SDN controllers come with their own Northbound APIs [16].

Conversely, the southbound API manages the transmission of control messages between the control and data planes, establishing communication with forwarding elements. Figure 2 illustrates the SDN architecture's structure. Each switch is equipped with OpenFlow (OF) capabilities, which include decision-making logic based on flow rules. OpenFlow configures the forwarding tables of individual switches [17], [18]. SDN supports programmability through various high-level APIs and languages, such as Procera [19], NetCore [20], and Frenetic [21], providing flexible options for developing diverse SDN applications [22]-[25].

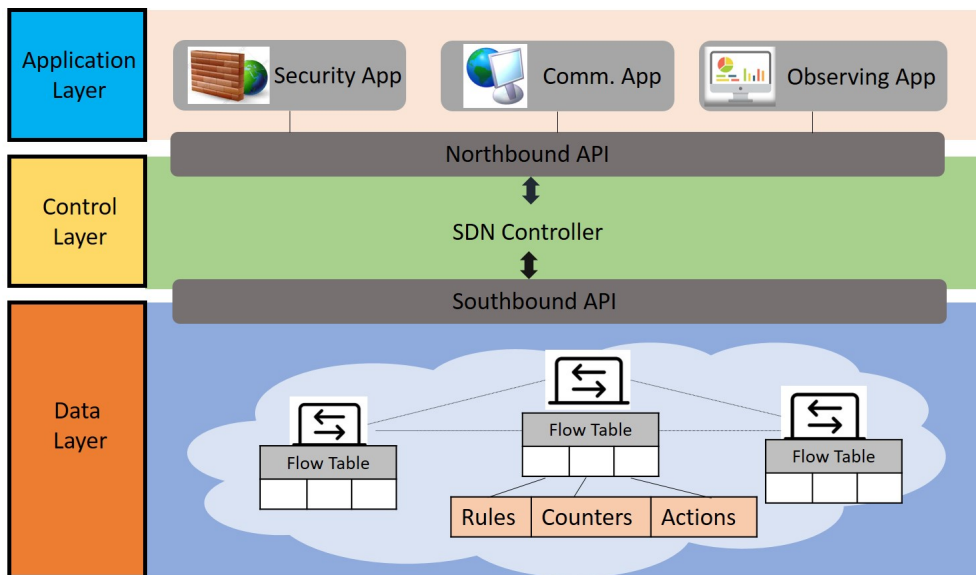


Figure 1. SDN architecture

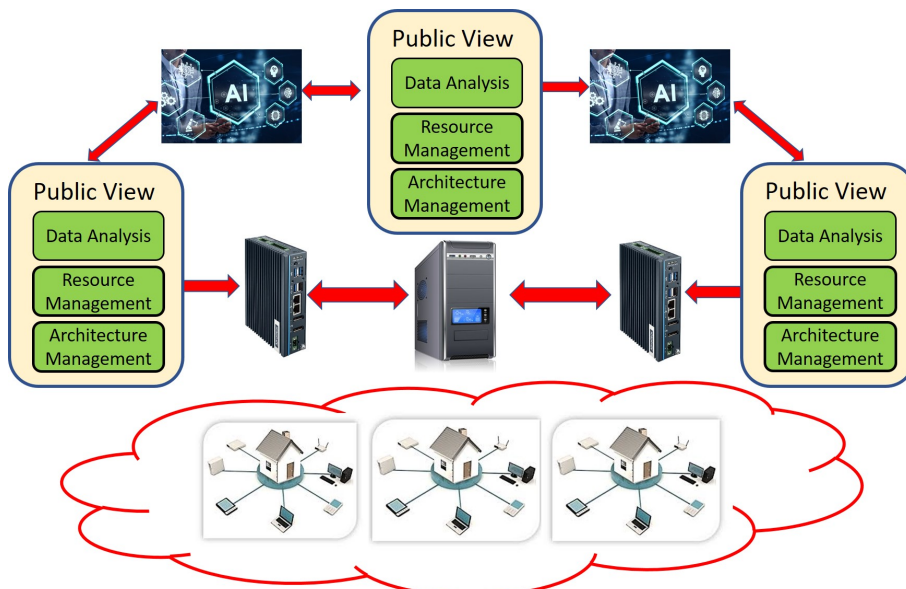


Figure 2. SDN public view

Artificial intelligence (AI), particularly machine learning (ML), significantly improves SDN load balancing through capabilities such as traffic pattern prediction, anomaly detection, and real-time dynamic adjustments. In AI-driven load balancing, traffic forecasting involves leveraging historical data to anticipate future loads and proactively allocate resources. Anomaly detection identifies unusual patterns indicative of potential issues like DDoS attacks or hardware malfunctions, while dynamic adjustments continuously optimize load distribution based on real-time conditions and predictions [26].

Supervised learning methods, such as neural networks (NNs) [27], support vector machines (SVMs) [28], decision trees (DTs) [29], ensemble approaches, and deep learning [30], are extensively applied in SDN. These techniques, including NNs, SVMs, and DTs, are used for tasks like intrusion detection and prevention [31], [32], controller placement [33], load balancing [34], performance forecasting [33], SLA enforcement, routing, and virtual machine placement [35]. NNs are particularly effective in applications such as intrusion detection [31], [36], load balancing, and SLA enforcement. For example, Chen and Yu [37] developed a neural network-based collaborative intrusion prevention architecture (CIPA), which used packet features like ICMP ratios, short/long packet ratios, and SYN/ACK ratios, demonstrating superior detection of DDoS and worm outbreaks with minimal computational overhead. Ge *et al.* [38] utilized multi-label classification with NNs for forecasting global network allocations, outperforming DT and LR models and achieving a two-thirds reduction in runtime. Luti [39] showed that NNs could effectively predict traffic demands in mobile networks with minimal optimality gaps. Similarly, Nandy *et al.* [40] implemented an NN-based SDN intrusion detection system achieving a 97.3% accuracy using the NSLKDD dataset. Gures *et al.* [41] improved load balancing performance with NNs, outperforming static Round Robin strategies by reducing network latency by 19.3%.

Shaji *et al.* [42] used a neural network approach incorporating the Levenberg-Marquardt algorithm for predicting SDN performance, yielding low mean squared error (MSE). Sarma *et al.* [43] applied long short-term memory (LSTM) networks for robust SLA enforcement in SDNs, which outperformed feedforward networks in predicting service-level breaches. Ashtari *et al.* [44] introduced a knowledge-defined networking approach using NNs for routing, achieving low MSE. SVMs also play a vital role in SDN for detecting DDoS attacks, traffic engineering, and routing. Long and Jinsong [45] presented an SVM-based solution for distinguishing legitimate traffic from DDoS attacks, which showed high accuracy and reduced false positives. Khedr *et al.* [46] applied SVMs with flow and packet-level features for DDoS detection with similarly high performance. SVMs have also been utilized for QoS-aware routing [47], traffic engineering [48], and other tasks [49].

Unsupervised learning techniques like K-means, self-organizing maps (SOM), hidden markov models (HMM), and restricted boltzmann machines (RBMs) are essential for traffic analysis and anomaly detection in SDNs. For example, K-means clustering and probabilistic models were used by Tan *et al.* [50] and Kanj *et al.* [51] for attack detection in SDN cloud environments. These methods utilized features such as session duration, packet rate, and TCP flag presence to group and analyze network traffic. Other researchers, like Chetouane and Karoui [52], compared various ML algorithms for DDoS detection, finding Naive Bayes to have the highest detection rate, while K-means exhibited faster response times. SOM has been effectively used for intrusion detection in SDN, though it can face challenges in matching traffic in flow tables [53]. DSOM, a distributed SOM approach, showed high accuracy in handling SDN performance issues under flooding attacks by training at each switch and clustering with K-means [54]. While supervised learning is popular for traffic classification, unsupervised methods are emphasized for their ability to detect patterns without annotated datasets. This characteristic allows for discovering new network applications, making unsupervised methods like K-means clustering a practical choice for models like $SDN_{CTRL_{ML}}$.

SDN distributed architectures such as ONOS [55] aim to enhance the SDN scalability and reliability by statically connecting controllers to switches. This setup creates a load imbalance issue due to the fluctuating switch traffic, which has a negative impact on the network performance. To address this problem, recent studies have employed switch migration mechanisms to redistribute switches among different controllers. Despite promising results, these approaches are complex and experience high latency due to the time overhead associated with the migration process.

The integration of the AI into SDN is promising to help in the load balancing problem. AI provides more intelligence and automated network management capabilities. Previous studies have highlighted ongoing efforts to incorporate AI into SDN [56]. Applying artificial intelligence to software-defined networks revolutionizes network management by infusing intelligent decision-making capabilities into the network infrastructure. Through AI, SDNs can dynamically adapt to changing network conditions, optimize traffic routing, and enhance security protocols in real-time. Machine learning algorithms analyze vast amounts of network data to

identify patterns, predict network behavior, and proactively mitigate potential issues. Additionally, AI-driven SDNs enable autonomous network management, reducing the need for manual intervention and streamlining operations. This integration not only improves network performance and reliability but also lays the groundwork for more efficient and resilient digital ecosystems. By leveraging AI, SDNs can intelligently distribute control plane tasks among multiple controllers, optimizing workload distribution and minimizing latency. This ensures efficient utilization of resources and enhances network scalability and resilience. Additionally, AI-driven load balancing enables SDNs to dynamically adjust controller assignments based on changing network conditions, ensuring optimal performance even during peak traffic periods or in the presence of network failures. By integrating AI into load balancing for distributed controllers, SDNs can achieve higher levels of automation, efficiency, and reliability in managing complex network environments. This study looked into the effects of AI machine learning approaches for load balancing. While previous studies investigated the impact of AI machine learning approaches for load balancing, they did not explicitly address its influence on response time and utilization.

This paper introduces $SDN_{CTRL_{ML}}$, a novel architecture for distributed controllers that uses AI machine learning approaches for load balancing. $SDN_{CTRL_{ML}}$ ensures load balancing among controllers through a scheduling layer that enables switch-controller association without fixed bindings. This allows switches to send messages to different controllers based on scheduling algorithms without interrupting switch re-association. This yields to maintaining network performance and minimizing the overall delays. $SDN_{CTRL_{ML}}$ provides low disturbance to the network performance. The scheduling process occurs seamlessly without interrupting switch re-association, ensuring service continuity and minimal network delays. $SDN_{CTRL_{ML}}$ is transparent to switches. $SDN_{CTRL_{ML}}$ does not introduce additional overhead for switches. This simplifies the deployment and configuration processes. It is able to assign a unique anycast controller IP address to switches that helps in simplifying the network deployment. $SDN_{CTRL_{ML}}$ provides high compatibility. It works with popular SDN distributed controller systems like ONOS [57] and Onix [58]. Additionally, $SDN_{CTRL_{ML}}$ also provides scalability. It allows dynamic adjustment of controller numbers to meet real-time network demands. It provides flexibility since it supports various scheduling algorithms. $SDN_{CTRL_{ML}}$ presents a promising solution for achieving low response times and high throughput. This ensures efficient load balancing among distributed controllers without compromising network performance or requiring additional switch configurations. Key points highlighted in the paper are:

- Uneven load distribution: existing distributed controller architectures suffer from uneven load distribution due to fixed associations between controllers and switches.
- Complexity and latency: prior attempts at solving this issue through switch migration introduce significant system complexity and network latency.
- Proposed solution: $SDN_{CTRL_{ML}}$
- Load balancing through scheduling layer: introduces a scheduling layer that intercepts flow requests from switches.
- Machine learning techniques: utilizes machine learning algorithms to allocate flows to different controllers based on selected scheduling algorithms.
- Seamless integration: operates without requiring additional modifications to standard SDN switches.
- Adaptability: supports various scheduling algorithms while minimizing disruption to services and network delay.

The main contributions of this work could be listed below:

- Prototype development: developed a prototype based on machine learning approaches that is integrated with different distributed controller systems.
- Experimental results: demonstrates the superiority of $SDN_{CTRL_{ML}}$ over reference model systems.
- Performance metrics: outperforms in terms of system throughput and response time.
- Complexity mitigation: achieves superior performance without introducing complexities associated with dynamic-binding controller systems.

This paper presents $SDN_{CTRL_{ML}}$, an innovative method that leverages machine learning to enhance load balancing. $SDN_{CTRL_{ML}}$ introduces a scheduling layer that utilizes machine learning algorithms to dynamically allocate flow requests to controllers. In contrast to earlier methods, $SDN_{CTRL_{ML}}$ seamlessly integrates with standard SDN switches and accommodates various scheduling algorithms, reducing disruptions

and network delays. Experimental results demonstrate that $SDN_{CTRL_{ML}}$ surpasses static-binding controller models while avoiding the complexities associated with dynamic-binding systems. The paper is organized as: section 1 has the background, problem statement and the proposed solution. Section 2 has the method used in this article. Section 3 is outlines the result and discussion. Finally, section 4 has the conclusion.

2. METHOD

2.1. Reference model

In this work, we used the BLAC [59] model as a reference model to be compared to this work. BLAC stands for bindingless architecture for distributed controllers (BLAC). BLAC achieves load balancing among controllers through a scheduling layer. The flow of requests are dispatched to different controllers through scheduling algorithms. The process is proceeded transparently with no extra modification required for off-the-shelf SDN switches. The scheduling layer is a crucial component in network systems that manages the allocation of resources and prioritizes tasks or data transmission. Its primary objective is to optimize the utilization of network resources and ensure efficient and reliable service delivery. To achieve this, the scheduling layer supports various scheduling algorithms, allowing for flexibility in adapting to different network requirements and traffic conditions. By employing different scheduling algorithms, such as round-robin, weighted fair queuing, or priority-based scheduling, the layer can dynamically allocate resources based on factors like priority, quality of service (QoS) requirements, available bandwidth, or latency constraints. This flexibility enables the scheduling layer to accommodate diverse network environments and application-specific needs.

A well-designed scheduling layer aims to minimize disruption of service and network delay. It achieves this by making intelligent decisions on resource allocation, considering factors such as the current state of the network, traffic conditions, and QoS requirements. By efficiently managing resource allocation and scheduling, the layer ensures that critical services receive appropriate resources without causing significant delays or interruptions for other network activities. The scheduling layer plays a vital role in optimizing resource utilization and ensuring reliable network performance. Its ability to support various scheduling algorithms and minimize disruptions allows for effective management of network resources while delivering efficient and uninterrupted services to users.

In this work, we chose BLAC as the reference model since it is a prototype that can work with various distributed controller systems and conduct experiments to demonstrate its efficacy. BLAC is chosen as the reference model, since according to its results, BLAC has proven to outperform its peers that use the static-binding controller system in terms of both system throughput and response time without the complexity of the dynamic-binding controller system.

2.2. Proposed model

The proposed model, named $SDN_{CTRL_{ML}}$, is a bindingless framework designed for distributed SDN controllers, leveraging the K-means machine learning algorithm to achieve load balancing. The primary objective of $SDN_{CTRL_{ML}}$ is to evenly distribute the workload within the SDN control plane. This is accomplished by integrating a scheduling layer into existing SDN architectures, thereby enhancing load balancing seamlessly. This approach eliminates the need for time-consuming switch migration processes and modifications to the existing switch configuration. To boost the flexibility and adaptability of the $SDN_{CTRL_{ML}}$ architecture, it incorporates multiple scheduling algorithms within the scheduling layer. This integration allows the architecture to effectively manage variations in communication demand across the network. The flowchart for $SDN_{CTRL_{ML}}$ is illustrated in Figure 3.

The $SDN_{CTRL_{ML}}$ architecture uses bindingless approach for distributed controllers. This ensures load balancing among various controllers. $SDN_{CTRL_{ML}}$ uses a scheduling layer that maintains a bindingless switch-controller association, eliminating the need for a switch to be exclusively associated with a single master controller. In this architecture, messages from switches are processed by multiple controllers. The scheduling layer intercepts switch requests and distributes them among different controllers using selected scheduling algorithms. This enables efficient load balancing that ensures that the workload is evenly distributed among the controllers. By decoupling switches from specific controllers, $SDN_{CTRL_{ML}}$ enhances the flexibility and adaptability of the architecture. Switches can interact with any available controller, enabling better utilization of resources and accommodating changes in the network conditions.

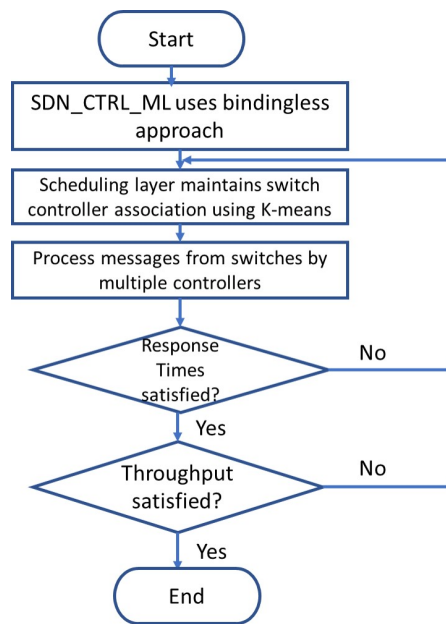


Figure 3. Flowchart for the $SDN_{CTRL_{ML}}$

The scheduling layer operates seamlessly during the switch re-association process resulting in minimal disruption to the network performance. This uninterrupted scheduling process ensures that services remain unaffected and network delays are kept to a minimum. Switches integrate with the $SDN_{CTRL_{ML}}$ framework without the need to any additional extensions to be installed. This ensures compatibility and also simplifies the deployment process. $SDN_{CTRL_{ML}}$ offers essential features that contribute to the establishment of a high-performance SDN network with low response times and increased throughput. $SDN_{CTRL_{ML}}$ assigns a single anycast controller IP address to all switches. This approach eliminates the need for individual controller IP addresses for each switch, making the network setup easier and more efficient. $SDN_{CTRL_{ML}}$ integrates with SDN distributed controller systems such as Onos [60] and Onix [61]. The high compatibility facilitates the interoperability and the ease of integration of $SDN_{CTRL_{ML}}$ with existing network environments that use these controller systems. $SDN_{CTRL_{ML}}$ offers the ability to dynamically update the number of controllers in real-time to meet the changing network demand. This scalability feature ensures that the network can flexibly adapt to varying traffic conditions and effectively allocate resources based on the evolving requirements.

The *switch adapter module* is responsible for handling the connections between switches and the scheduling instance. Its primary function is to listen on an anycast IP address, which allows it to receive requests from neighboring switches. Similarly, the *controller adapter module* handles the connections between the scheduling instance and the controllers. It facilitates communication and data exchange between the scheduling layer and the controllers. Additionally, the controller adapter collects controller-related information (CPU utilization) to assist in the scheduling decisions and the load balancing process. The *scheduling module* is responsible for routing messages between switches and controllers based on the underlying scheduling algorithms. To achieve controller load balancing, $SDN_{CTRL_{ML}}$ integrates a scheduling layer into an SDN network. The scheduling layer accommodates multiple scheduling instances. $SDN_{CTRL_{ML}}$ exhibits flexibility by supporting a diverse array of scheduling algorithms, including randomized scheduling [62] and round robin scheduling [63]. In Figure 4, we depict a single instance for simplicity, positioned between the switches and controllers. The scheduling module determines the appropriate controller(s) to handle incoming switch requests and ensures efficient load balancing. $SDN_{CTRL_{ML}}$ leverages anycast technology to simplify configuration and reduce network latency, contributing to a streamlined and efficient network setup.

$SDN_{CTRL_{ML}}$ employs a bindingless SDN architecture to resolve the issue of controller load imbalance caused by the traditional switch-controller binding. In this model, switches are linked to a scheduling layer rather than being directly attached to controllers. The scheduling layer facilitates transparent forwarding of packets from the switches to any available controller. The logical sequence of packet dispatching in $SDN_{CTRL_{ML}}$, as illustrated in Figure 5, is as follows: the switch adapter first receives and processes the pack-

ets from the switches, then forwards the Packet-In requests to the scheduling module. The scheduling module, using its scheduling algorithms, makes a decision and instructs the controller adapter to send the request to a designated controller. The controller adapter captures the Packet-Out responses from the controllers and routes them back to the scheduling module. With this information, the scheduling module identifies the target switch that should receive the response. This bindingless structure supports flexible and dynamic packet routing between switches and controllers, eliminating the limitations of conventional switch-controller binding and efficiently mitigating controller load imbalance. Figure 5 illustrates the design of the $SDN_{CTRL_{ML}}$ scheduler and details the packet flow from switches through the $SDN_{CTRL_{ML}}$ scheduling layer to the controllers.

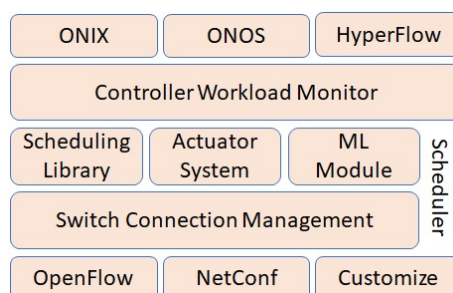


Figure 4. $SDN_{CTRL_{ML}}$ scheduling module

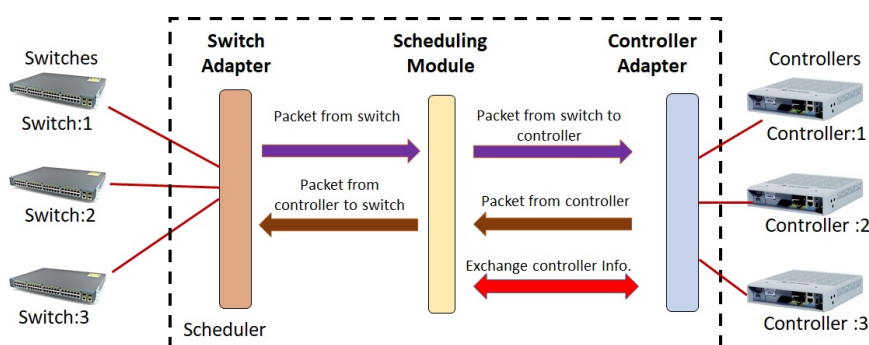


Figure 5. $SDN_{CTRL_{ML}}$ operation

The proposed model, $SDN_{CTRL_{ML}}$, has the following assumptions:

- Assumed that all requests have the same workload
- All controllers have the same capacity
- CPU utilization is the load on a controller
- CPU is the throughput bottleneck of the controller
- CPU load is proportional to the request arrival rate
- Placed schedulers near the controllers
- Network delay between schedulers and controllers equals to zero
- Used load balancing per flow of data holding the same characteristics

$SDN_{CTRL_{ML}}$ ensures load balancing while maintaining information consistency across controllers. It utilizes the widely adopted network topology consistency maintenance mechanism inherent in distributed controller systems. For instance, in the ONOS system, each controller preserves a coherent global network view within its memory through a notification-based replication scheme. This approach allows $SDN_{CTRL_{ML}}$ to route packets to various controllers without compromising the accuracy of the results. These packets, known as independent requests, are processed based on the synchronized network topology that is accessible to all controllers.

However, future applications like ONOS PIM may introduce dependent requests that require controller-specific private information. To handle this, the scheduling layer in $SDN_{CTRL_{ML}}$ can apply exception rules.

For example, dependent packets can be consistently routed to the same controller to maintain processing accuracy. Alternatively, the mastership model can be upheld, applying exceptions only to packets that are universally processable by any controller, such as ARP messages. These strategies help ensure that both independent and dependent requests are processed correctly, maintaining the consistency of information among controllers.

In the $SDN_{CTRL_{ML}}$ architecture, switches connect to the scheduling layer using the anycast technique. This technique offers several key advantages such as (1) simplicity: where the used anycast technique allows multiple scheduling instances within a network to share the same anycast IP address. As a result, switch configurations become simplified and uniform. By leveraging the anycast technique, switches automatically connect to the scheduling instance that is closest to them in terms of proximity. This proximity-based connection reduces network latency. Additionally, the scheduling instances in our architecture are cost-effective and easy to create. They can be strategically positioned to achieve load balancing among them, ensuring a balanced distribution of network traffic and minimizing network latency. Figure 4 has the details about the implementation of the $SDN_{CTRL_{ML}}$ scheduler module along with the controller association. The scheduler has the controller workload monitor, scheduling library that is using weighted round robin, actuator system that takes the action based on the load balancing status.

Algorithm ?? has the algorithm used for $SDN_{CTRL_{ML}}$. $SDN_{CTRL_{ML}}$ utilizes machine learning to improve load balancing in SDN networks. The algorithm dynamically assigns flow requests to controllers based on the predicted load and network conditions. It leverages machine learning to dynamically assign flow requests to controllers, thus optimizing load distribution.

Algorithm 1. ML-based load balancing for SDN- $SDN_{CTRL_{ML}}$

```

1: Input: Flow requests  $F$ , Controllers  $C$ , Historical load data  $H$ 
2: Output: Assignment of flow requests to controllers
3: Train machine learning model  $SDN_{CTRL_{ML}}$  using K-means machine learning to cluster historical load data  $H$ 
4: for each flow request  $f \in F$  do
5:   Predict load on each controller  $c \in C$  using  $ML$ 
6:   Select controller  $c_{\min} \leftarrow \arg \min_{c \in C} \text{PredictedLoad}(c)$ 
7:   Assign flow request  $f$  to controller  $c_{\min}$ 
8:   Update load data  $H$  with new load on  $c_{\min}$ 
9: end for
10: return Assignment of flow requests

```

The machine learning model $SDN_{CTRL_{ML}}$ is trained using K-means historical load data H . This data includes information about previous flow requests and the resulting load on each controller. Various features such as time of day, flow characteristics, and network conditions can be used for training. For each new flow request f , the trained model ML predicts the load on each controller $c \in C$. The prediction is based on the current state of the network and the characteristics of the flow request. The algorithm dynamically assigns each flow request to the controller with the minimum predicted load. This ensures that the load is balanced across all controllers, optimizing network performance. The proposed algorithm provides an effective method for improving load balancing in SDN networks using machine learning.

3. RESULTS AND DISCUSSION

In this section, we provide the evaluation of $SDN_{CTRL_{ML}}$. Initially, we elaborate on the experimental testbed utilized for gauging the response time and throughput of $SDN_{CTRL_{ML}}$. Subsequently, we undertake a comparative analysis between $SDN_{CTRL_{ML}}$ and the static binding controller system ONOS to underscore the efficacy of our architecture.

3.1. Experiment setup

The experimental setup is established on Mininet [64], a network emulator that is widely employed for exploring the functionalities of novel SDN components. However, its use for performance evaluation is limited due to emulation overhead. To alleviate this, we adopt three strategies inspired by [64]. Initially, we modify Mininet's source code to create GRE tunnels between hosts, thereby reducing the traffic in the emulated

data plane. Secondly, we reconfigure Mininet to enable vSwitches to directly inject Packet-In messages to controllers, bypassing the data plane, and disable Flow-Mod message transmission to lessen switches' flow table overhead. These alterations render the performance evaluation on Mininet a reasonable reflection of a real network [64]. Experiments are conducted on a testbed comprising five physical machines equipped with quad-core 4 GHz Intel Core i7 processors and 16 GB DDR3 RAM, running the latest 1.6 stable version of ONOS [60]. Demonstrating the feasibility and efficiency of $SDN_{CTRL_{ML}}$, we examine various applications such as ACL, reactive routing, RabbitMQ, and reactive forwarding [60]. Results exhibit consistent trends across these applications, focusing on performance gain rather than absolute values. Therefore, we present results pertaining to reactive forwarding for illustration purposes. The system's performance is tested with different numbers of schedulers, revealing no significant differences in $SDN_{CTRL_{ML}}$ effectiveness. This is attributed to the controller, rather than the scheduler, being the primary performance bottleneck.

3.2. Discussion

Figure 6 has the response time trends for different scheduling algorithms in $SDN_{CTRL_{ML}}$ and ONOS. In ONOS, the response time remains nearly constant (around 1.5 ms) at arrival rates below 27,000 pkt/s. As the rate exceeds the threshold, the response time significantly escalates. Conversely, with $SDN_{CTRL_{ML}}$, scheduling algorithms exhibit more consistent response times, even at higher arrival rates. Notably, load distribution among controllers significantly affects response times, with $SDN_{CTRL_{ML}}$ demonstrating improved workload balancing, thus reducing response times compared to ONOS.

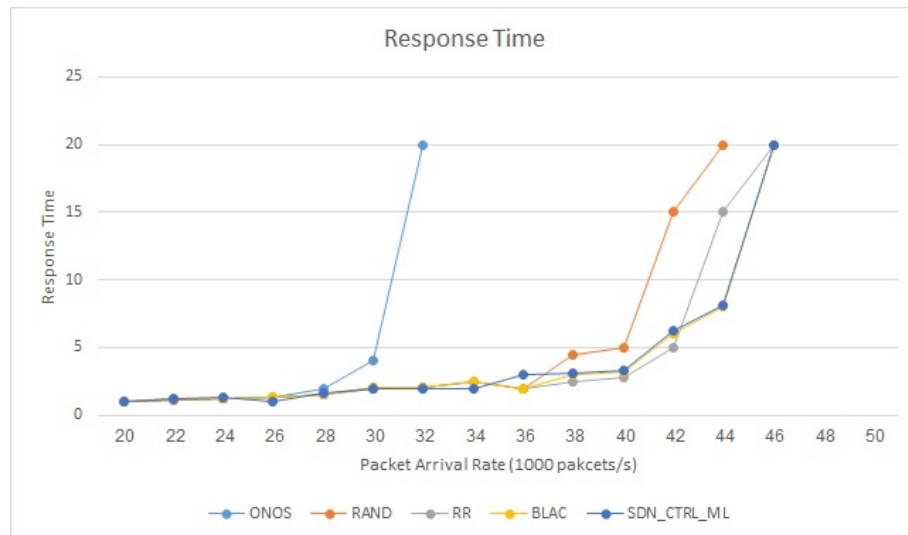


Figure 6. $SDN_{CTRL_{ML}}$ comparing $SDN_{CTRL_{ML}}$ response time to other approaches

Figure 6 illustrates that ONOS's response time remains stable at approximately 1.5 ms for arrival rates below 27,000 packets per second (pkt/s). Once the arrival rate surpasses this point, the response time rises significantly. For randomized scheduling and round-robin scheduling methods, the response time stays under 6 ms as long as the arrival rate is less than 38,000 pkt/s. The response time starts to increase sharply only when the arrival rate exceeds 42,000 pkt/s. Similarly, with improved randomized scheduling and weighted round-robin algorithms, the response time remains close to 5.5 ms even when the arrival rate goes beyond 3,000 pkt/s. To put this into perspective, imagine each controller as an M/M/1 queue. When the service rate of the queue exceeds the arrival rate, the service time (or response time) remains low. However, if the arrival rate greatly exceeds the service rate, the queue length grows quickly, leading to a sharp increase in response time. This explains the rapid escalation in response time as the arrival rate exceeds a certain threshold.

Figure 7 presents a comparison of throughput between $SDN_{CTRL_{ML}}$ and BLAC across different scheduling algorithms, where throughput is measured as CPU utilization. The results indicate that the overall system throughput scales linearly with the number of controllers. Implementing $SDN_{CTRL_{ML}}$ significantly boosts system throughput compared to BLAC. Advanced scheduling algorithms, such as improved randomized and weighted round-robin, show higher throughput compared to simpler scheduling methods.

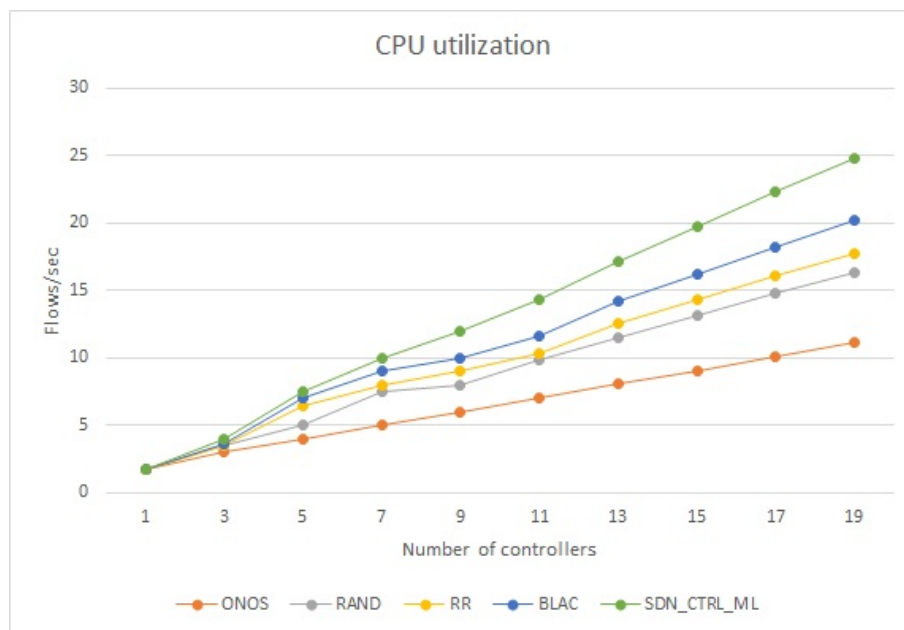


Figure 7. Throughput-comparing $SDN_{CTRL_{ML}}$ CPU utilization to other approaches

Response times vary depending on the scheduling algorithm used, influenced by how requests are distributed among controllers. In BLAC, the static nature of connections can lead to an uneven workload distribution, causing some controllers to be overloaded while others are underutilized. This imbalance results in longer response times, even at moderate overall system workloads. In contrast, $SDN_{CTRL_{ML}}$ achieves a more balanced workload distribution, improving the system's ability to manage requests and reducing response times.

To evaluate performance, we measure both CPU utilization of all controllers and response times for different scheduling algorithms in $SDN_{CTRL_{ML}}$. The impact of the number of probes on response time is particularly noteworthy: using probes substantially lowers response times, especially at higher Packet-In arrival rates. However, excessive probes can add message overhead, which may negatively affect performance. Maintaining a moderate probe count—such as two—proves beneficial, as shown in Figure 8.

When request demand is low, the number of probes per request has minimal impact on performance. However, as the volume of incoming requests rises, the number of probes used becomes more critical. Figure 8 illustrates that using probes significantly reduces response time compared to random dispatching. For example, at an arrival rate of 41,000 packets per second, employing probes reduces response time by more than 50% relative to random dispatching. The figure also shows diminishing performance gains beyond two probes at low request volumes, and a potential decline in performance under high load due to increased message overhead with excessive probes. Therefore, in our experiments, we use two probes for enhanced randomized scheduling.

Figure 9 demonstrates that utilizing request buffering enhances performance, yet negligible improvement is observed beyond buffering two requests. Over-burdening with buffered requests can deteriorate system performance, especially under high request demand. Therefore, maintaining a request-buffering number of 2 is chosen for implementation throughout our experiments. Figure 9 shows that the enhanced randomized scheduling technique decreases response time further by using batch assignment. The red line depicts the “power of two choices” method, where requests are dispatched without buffering. In a network with seven controllers, algorithms employing batch assignment outperform the “power of two choices” technique because the schedulers can more effectively find less loaded controllers by buffering requests. However, buffering more than two requests rarely reduces response time. In fact, system performance can decline when additional requests are buffered under high demand. This is because, as the number of incoming requests grows, the probability of sampling a lightly loaded controller decreases, and response time worsens with increased buffering. Additionally, high demand often causes traffic congestion, resulting in longer buffering times.

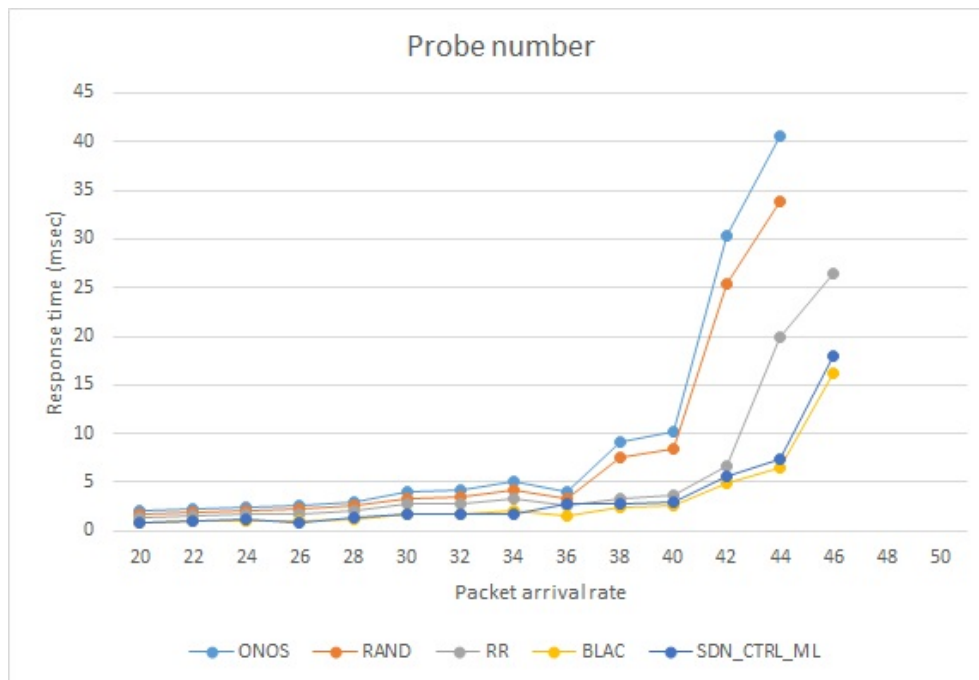


Figure 8. Comparing probe numbers for $SDN_{CTRL_{ML}}$ to other approaches

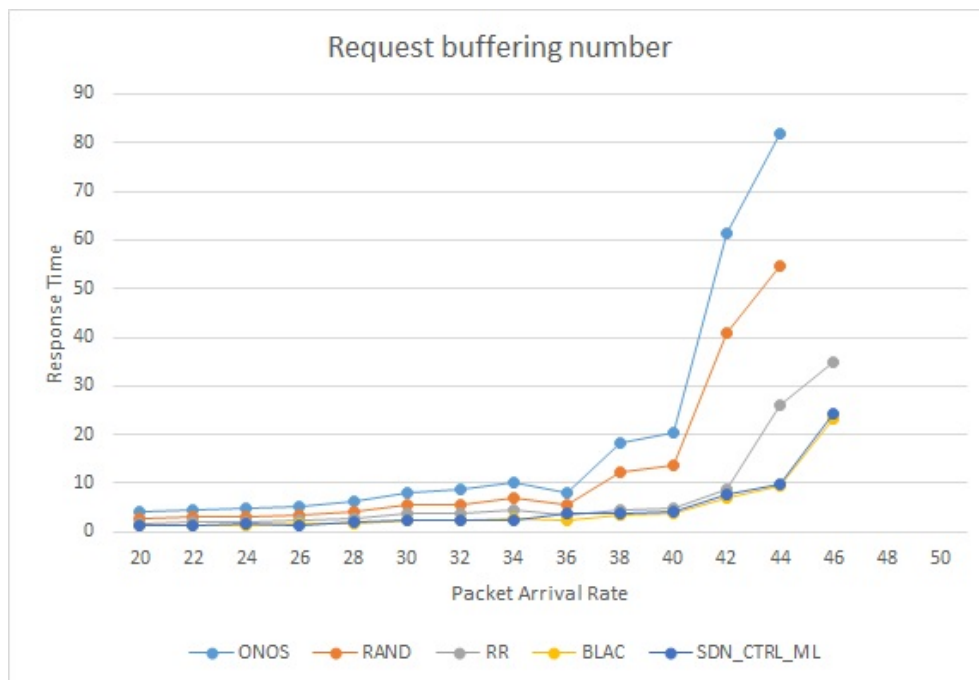


Figure 9. Comparing request buffering number for $SDN_{CTRL_{ML}}$ to other approaches

We create traffic at a rate of 80,000 packets per second as depicted in Figure 10. It is clear that optimizing the workload distribution for the controller enhances system performance. Notably, disparities in load distribution are evident across various scheduling algorithms; three controllers are strained, operating at over 90% CPU utilization, while two controllers are underutilized, hovering around 45% CPU utilization. Consequently, this imbalance results in response times exceeding 20 milliseconds. In contrast, the response time for

$SDN_{CTRL_{ML}}$ is notably shorter, approximately 9 milliseconds, across different scheduling algorithms. Figure 10 demonstrates that system performance improves with balanced controller workloads. When using BLAC alone, the load imbalance is significant: three controllers become overloaded while two controllers are under-utilized, leading to a response time exceeding 15 milliseconds. In contrast, $SDN_{CTRL_{ML}}$ achieves a much lower response time, approximately 4 milliseconds, across various scheduling algorithms compared to BLAC.

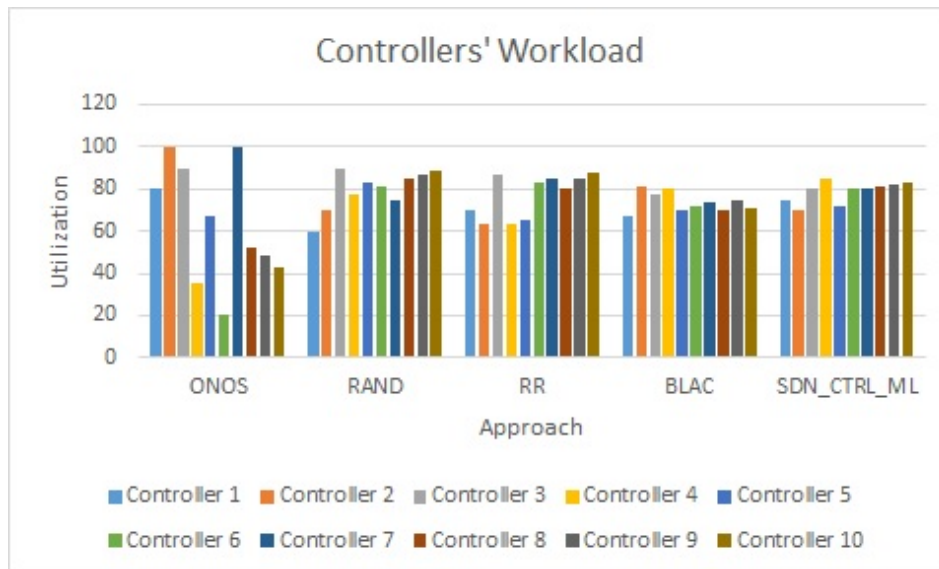


Figure 10. Comparing controller's workload for $SDN_{CTRL_{ML}}$ to other approaches

4. CONCLUSION

The paper addresses the challenge of uneven load distribution in SDN caused by fixed associations between controllers and switches. It highlights the complexity and latency issues introduced by existing solutions that use switch migration for load balancing. To tackle this, the paper introduces $SDN_{CTRL_{ML}}$, a novel approach employing a scheduling layer powered by machine learning to balance controller loads seamlessly. This layer intercepts switch flow requests and allocates them to controllers using machine learning-based scheduling algorithms. Notably, this process doesn't require modifications to standard SDN switches and minimally disrupts services while maintaining low network delay. The paper includes a prototype integrated with various distributed controller systems and validates $SDN_{CTRL_{ML}}$ through experiments, demonstrating its superior performance in system throughput and response time compared to static-binding and reference model systems, all without the complexities associated with dynamic-binding controller approaches. Recent observations indicate that the $SDN_{CTRL_{ML}}$ outperforms BLAC reference model by 13% in terms of response time and 14% in terms of utilization. A future extension of this work is to use more complicated machine learning approaches and compare the results to the current approach.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Wael Hosny Fouad Aly	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓
Hassan Kanj		✓				✓		✓	✓	✓	✓	✓	✓	
Nour Mostafa	✓		✓	✓		✓			✓		✓	✓	✓	
Zakwan Al-Arnaout	✓		✓	✓		✓			✓		✓		✓	
Hassan Harb					✓		✓			✓		✓		✓

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal Analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review & Editing

Vi : Visualization

Su : Supervision

P : Project Administration

Fu : Funding Acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES




- [1] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han, and C. S. Hong, "Network slicing: Recent advances, taxonomy, requirements, and open research challenges," *IEEE Access*, vol. 8, pp. 36 009–36 028, 2020, doi: 10.1109/ACCESS.2020.2975072.
- [2] S. Ahmad and A. H. Mir, "Scalability, consistency, reliability and security in sdn controllers: a survey of diverse sdn controllers," *Journal of Network and Systems Management*, vol. 29, pp. 1–59, 2021, doi: 10.1007/s10922-020-09575-4.
- [3] I. Alam *et al.*, "A survey of network virtualization techniques for internet of things using sdn and nfv," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–40, 2020, doi: 10.1145/3379444.
- [4] K. Nisar *et al.*, "A survey on the architecture, application, and security of software defined networking: Challenges and open issues," *Internet of Things*, vol. 12, p. 100289, 2020, doi: 10.1016/j.iot.2020.100289.
- [5] B. Almadani, A. Beg, and A. Mahmoud, "Dsf: A distributed sdn control plane framework for the east/west interface," *IEEE Access*, vol. 9, pp. 26 735–26 754, 2021, doi: 10.1109/ACCESS.2021.3057690.
- [6] H. Kanj and J.-M. Flaus, "A simulation approach for risk modeling and analysis based on multi-agents," in *ESREL 2015, 25th European Safety and Reliability Conference*, 2015, doi: 10.1201/b19094-514.
- [7] S. Prabakaran *et al.*, "Predicting attack pattern via machine learning by exploiting stateful firewall as virtual network function in an sdn network," *Sensors*, vol. 22, no. 3, p. 709, 2022, doi: 10.3390/s22030709.
- [8] A. Abuarqoub, "A review of the control plane scalability approaches in software defined networking," *Future Internet*, vol. 12, no. 3, p. 49, 2020, doi: 10.3390/fi12030049.
- [9] S. H. A. Kazmi, F. Qamar, R. Hassan, K. Nisar, and B. S. Chowdhry, "Survey on joint paradigm of 5g and sdn emerging mobile technologies: Architecture, security, challenges and research directions," *Wireless Personal Communications*, vol. 130, no. 4, pp. 2753–2800, 2023, doi: 10.1007/s11277-023-10402-7.
- [10] A. M. Abdelrahman *et al.*, "Software-defined networking security for private data center networks and clouds: Vulnerabilities, attacks, countermeasures, and solutions," *International Journal of Communication Systems*, vol. 34, no. 4, p. e4706, 2020, doi: 10.1002/dac.4706.
- [11] W. Aly, "LBFTFB fault tolerance mechanism for software defined networking," *2017 International Conference On Electrical And Computing Technologies And Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, 2017, pp. 1-5, doi: 10.1109/ICECTA.2017.8251995.
- [12] W. Aly and Y. Kotb, "Towards SDN fault tolerance using petri-nets," *2018 28th International Telecommunication Networks And Applications Conference (ITNAC)*, Sydney, NSW, Australia, 2018, pp. 1-3, doi: 10.1109/ATNAC.2018.8615188.
- [13] W. Aly, H. Kanj, S. Alabed, N. Mostafa, and K. Safi, "Dynamic feedback versus varna-based techniques for SDN controller placement problems," *Electronics*, vol. 11, no. 14, 2022, doi: 10.3390/electronics11142273.
- [14] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (sdn) data plane security: issues, solutions, and future directions," *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, pp. 341–387, 2020, doi: 10.1007/978-3-030-22277-2_14.
- [15] S. Ahmad and A. H. Mir, "SDN interfaces: protocols, taxonomy and challenges," *International Journal of Wireless and Microwave Technologies (IJWMT)*, vol. 12, no. 2, pp. 11–32, 2022, doi: 10.5815/ijwmt.2022.02.02.
- [16] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *Journal of Network and Computer Applications*, vol. 156, p. 102563, 2020, doi: 10.1016/j.jnca.2020.102563.
- [17] M. Nasir and M. Ali, "Qualitative analysis of hybrid flow installation mechanism in software defined networks (SDN)," *Wireless Personal Communications*, vol. 116, pp. 3413–3464, 2021, doi: 10.1007/s11277-020-07859-1.
- [18] N. McKeown *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

- [19] R. Kumar, A. Aggarwal, K. Handa, P. Soni, and M. Kumar, "Software-defined networks and its applications," *Software Defined Networks: Architecture and Applications*, pp. 63–96, 2022, doi: 10.1002/9781119857921.ch3.
- [20] O. Michel, R. Bifulco, G. Retvari, and S. Schmid, "The programmable data plane: Abstractions, architectures, algorithms, and applications," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–36, 2021, doi: 10.1145/3447868.
- [21] Y. Maleh, Y. Qasmaoui, K. El Gholami, Y. Sadqi, and S. Mounir, "A comprehensive survey on sdn security: threats, mitigations, and future directions," *Journal of Reliable Intelligent Environments*, vol. 9, no. 2, pp. 201–239, 2023, doi: 10.1007/s40860-022-00171-8.
- [22] W. H. F. Aly, "A new controller placement technique using colored petri-nets modelling for sdns," in *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCLOUD/SocialCom/SustainCom)*, IEEE, pp. 941–947, 2020, doi: 10.1109/ISPA-BDCLOUD-SocialCom-SustainCom51426.2020.00144.
- [23] W. H. F. Aly, "A novel controller placement technique for sdns using petri-nets," in *2020 PMU International Conference on Industrial Revolution 4.0 in Computing, Mobility, and Manufacturing*, *IEEE Access*, 2020, doi: 10.37394/23206.2020.19.65.
- [24] W. H. F. Aly, H. Kanj, N. Mostafa, and S. Alabed, "Feedback arma models versus bayesian models towards securing openflow controllers for SDNs," *Electronics*, vol. 11, no. 9, p. 1513, 2022, doi: 10.3390/electronics11091513.
- [25] G. Treider, "Investigation of the gap between traditional ip network security management and the adoption of automation techniques and technologies to network security," *Master's thesis in IT Security Management. Dept. of Comp. Science, Norwegian University of Science and Technology*, Norwegian, 2023.
- [26] H. Kanj, Y. Kotb, M. Alakkoumi, and S. Kanj, "Dynamic decision making process for dangerous good transportation using a combination of topsis and ahp methods with fuzzy sets," *IEEE Access*, vol. 12, pp. 40450–40479, 2024, doi: 10.1109/ACCESS.2024.3372852.
- [27] B. Sudharsan, P. Patel, J. G. Breslin, and M. I. Ali, "Enabling machine learning on the edge using sram conserving efficient neural networks execution approach," in *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part V 21*, 2021, pp. 20–35, doi: 10.1007/978-3-030-86517-7_2.
- [28] D. M. Abdullah and A. M. Abdulazeez, "Machine learning applications based on svm classification a review," *Qubahan Academic Journal*, vol. 1, no. 2, pp. 81–90, 2021, doi: 10.48161/qaj.v1n2a50.
- [29] M. Latah and L. Toker, "A novel intelligent approach for detecting dos flooding attacks in software-defined networks," *International Journal of Advances in Intelligent Informatics*, vol. 4, no. 1, pp. 11–20, 2018, doi: 10.26555/ijain.v4i1.138.
- [30] M. M. Ali, B. K. Paul, K. Ahmed, F. M. Bui, J. M. Quinn, and M. A. Moni, "Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison," *Computers in Biology and Medicine*, vol. 136, p. 104672, 2021, doi: 10.1016/j.combiomed.2021.104672.
- [31] M. Al-Janabi, M. A. Ismail, and A. H. Ali, "Intrusion detection systems, issues, challenges, and needs," *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 560–571, 2021, doi: 10.2991/ijcis.d.210105.001.
- [32] O. H. Abdulganiyu, T. Ait Tchakoucht, and Y. K. Saheed, "A systematic literature review for network intrusion detection system (IDS)," *International Journal of Information Security*, vol. 22, no. 5, pp. 1125–1162, 2023, doi: 10.1007/s10207-023-00682-2.
- [33] M. W. Wang, J. M. Goodman, and T. E. Allen, "Machine learning in predictive toxicology: recent applications and future directions for classification models," *Chemical Research in Toxicology*, vol. 34, no. 2, pp. 217–239, 2020, doi: 10.1021/acs.chemrestox.0c00316.
- [34] M. Rizvi, "Powering the future: Unleashing the potential of machine learning for intelligent energy forecasting and load prediction in smart grids," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 6, no. 1, 2024, doi: 10.56726/IRJMETS48655.
- [35] N. Hogade and S. Pasricha, "A survey on machine learning for geo-distributed cloud data center managements," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 1, pp. 15–31, 2022, doi: 10.48550/arXiv.2205.08072.
- [36] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2451–2479, 2020, doi: 10.1109/TNSM.2020.3016246.
- [37] X.-F. Chen and S.-Z. Yu, "CIPA: A collaborative intrusion prevention architecture for programmable network and SDN," *Computers & Security*, vol. 58, pp. 1–19, 2016, doi: 10.1016/j.cose.2015.11.008.
- [38] R. Ge, R. Zhang, and P. Wang, "Prediction of chronic diseases with multi-label neural network," *IEEE Access*, vol. 8, pp. 138210–138216, 2020, doi: 10.1109/ACCESS.2020.3011374.
- [39] T. W. Luti, "Multi factor based mobile voice and data traffic forecasting using artificial neural networks," Ph.D. dissertation, University of Nairobi, 2022.
- [40] S. Nandy, M. Adhikari, M. A. Khan, V. G. Menon, and S. Verma, "An intrusion detection mechanism for secured iomt framework based on swarm-neural network," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 5, pp. 1969–1976, 2021, doi: 10.1109/JBHI.2021.3101686.
- [41] E. Gures, I. Shayea, M. Ergen, M. H. Azmi, and A. A. El-Saleh, "Machine learning-based load balancing algorithms in future heterogeneous networks: A survey," *IEEE Access*, vol. 10, pp. 37689–37717, 2022, doi: 10.1109/ACCESS.2022.3161511.
- [42] N. S. Shaji, T. Jain, R. Muthalagu, and P. M. Pawar, "Deep-discovery: Anomaly discovery in software-defined networks using artificial neural networks," *Computers & Security*, vol. 132, p. 103320, 2023, doi: 10.1016/j.cose.2023.103320.
- [43] H. K. D. Sarma, "A survey on machine learning and deep learning based quality of service aware protocols for software defined networks," *Authorea Preprints*, 2023, doi: 10.36227/techrxiv.16950574.v1.
- [44] S. Ashtari, I. Zhou, M. Abolhasan, N. Shariati, J. Lipman, and W. Ni, "Knowledge-defined networking: Applications, challenges and future work," *Array*, vol. 14, p. 100136, 2022, doi: 10.1016/j.array.2022.100136.
- [45] Z. Long and W. Jinsong, "A hybrid method of entropy and ssae-svm based ddos detection and mitigation mechanism in SDN," *Computers & Security*, vol. 115, p. 102604, 2022, doi: 10.1016/j.cose.2022.102604.
- [46] W. I. Khedr, A. E. Gouda, and E. R. Mohamed, "FMDADM: A multi-layer ddos attack detection and mitigation framework using machine learning for stateful sdn-based iot networks," *IEEE Access*, vol. 11, pp. 28934–28954, 2023, doi: 10.1109/ACCESS.2023.3260256.




- [47] W. Zheng *et al.*, "Application-aware qos routing in SDNss using machine learning techniques," *Peer-to-Peer Networking and Applications*, vol. 15, no. 1, pp. 529–548, 2022, doi: 10.1007/s12083-021-01262-8.
- [48] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, "SADM-SDNC: security anomaly detection and mitigation in software-defined networking using c-support vector classification," *Computing*, vol. 103, no. 4, pp. 641–673, 2021, doi: 10.1007/s00607-020-00866-x.
- [49] A. I. Owusu and A. Nayak, "An intelligent traffic classification in SDN-IoT: A machine learning approach," in *2020 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, IEEE, 2020, pp. 1–6, doi: 10.1109/BlackSeaCom48709.2020.9235019.
- [50] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, and Y. Deng, "A new framework for ddos attack detection and defense in sdn environment," *IEEE Access*, vol. 8, pp. 161 908–161 919, 2020, doi: 10.1109/ACCESS.2020.3021435.
- [51] S. Kanj, F. Abdallah, and T. Deneux, "Purifying training data to improve performance of multi-label classification algorithms," in *2012 15th International Conference on Information Fusion*, IEEE, pp. 1784–1791, 2012.
- [52] A. Chetouane and K. Karoui, "A survey of machine learning methods for DDoS threats detection against SDN," in *International Workshop on Distributed Computing for Emerging Smart Networks*, Springer, 2022, pp. 99–127, 10.1007/978-3-030-99004-6_6.
- [53] P. Harikrishna and A. Amuthan, "SDN-based DDoS attack mitigation scheme using convolution recursively enhanced self organizing maps," *Sādhanā*, vol. 45, no. 1, p. 104, 2020, doi: 10.1007/s12046-020-01353-x.
- [54] K. Savitha and C. Chandrasekar, "A hybrid intrusion detection model for vanet using sdn and growing hierarchical self-organizing maps," *Webology*, vol. 18, no. 5, 2021.
- [55] H. K. Ravuri, M. T. Vega, J. van der Hooft, T. Wauters, and F. De Turck, "A scalable hierarchically distributed architecture for next-generation applications," *Journal of Network and Systems Management*, vol. 30, pp. 1–32, 2022, doi: 10.1007/s10922-021-09618-4.
- [56] R. Etengu, S. C. Tan, L. C. Kwang, F. M. Abbou, and T. C. Chuah, "AI-assisted framework for green-routing and load balancing in hybrid software-defined networking: Proposal, challenges and future perspective," *IEEE Access*, vol. 8, pp. 166 384–166 441, 2020, doi: 10.1109/ACCESS.2020.3022291.
- [57] A. Singh, N. Kaur, and H. Kaur, "Extensive performance analysis of opendaylight (odl) and open network operating system (onos) sdn controllers," *Microprocessors and Microsystems*, vol. 95, p. 104715, 2022, doi: 10.1016/j.micpro.2022.104715.
- [58] S. Bhardwaj, S. N. Panda, Muskaan and P. Datta, "Comparison and Performance Evaluation of Software-Defined Networking Controllers," *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, Pune, India, 2020, pp. 276–281, doi: 10.1109/ESCI48226.2020.9167554
- [59] E. A. Turner, S. P. Harrell, and T. Bryant-Davis, "Black love, activism, and community (blac): The blac model of healing and resilience," *Journal of Black Psychology*, vol. 48, no. 3-4, pp. 547–568, 2022, doi: 10.1177/0095798421101836.
- [60] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 1–6, doi: 10.1145/2620728.2620744.
- [61] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *OSDI*, vol. 10, no. 1, p. 6, 2010, doi: 10.5555/1924943.1924968.
- [62] B. Sellami, A. Hakiri, S. B. Yahia, and P. Berthou, "Energy-aware task scheduling and offloading using deep reinforcement learning in SDN-enabled iot network," *Computer Networks*, vol. 210, p. 108957, 2022, doi: 10.1016/j.comnet.2022.108957.
- [63] Y. A. H. Omer, A. B. A. Mustafa, and A. G. Abdalla, "Performance analysis of round robin load balancing in SDN," in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, IEEE, 2021, pp. 1–5, doi: 10.1109/ICCCEEE49695.2021.9429662.
- [64] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 1–6, 2010, doi: 10.1145/1868447.1868466.

BIOGRAPHIES OF AUTHORS







Wael Hosny Fouad Aly    has received his Ph.D. degree at the University of Western Ontario in Canada in 2006. He is a Professional Engineer of Ontario P.Eng. (Canada). He is currently working as a Professor of Computer Engineering at the College of Engineering and technology at the American University of the Middle East in Kuwait since 2016. His research interests include SDN networking, distributed systems, optical burst switching (OBS), wireless sensor networks (WSN), differentiated services, and multi-agent systems. He is a senior member of the IEEE and the IEEE Computer Society. He can be contacted at email: wael.aly@aum.edu.kw.







Hassan Kanj    received a M.Sc. degree in Software Engineering from the Lebanese University (LU) in 2011, and in Computer Engineering from the University of Technology of Compiègne (UTC) in 2013. He pursued his Ph.D. studies in Computer Engineering at the University of Grenoble Alpes (UGA) in France. In 2016, he defended his dissertation entitled: Contribution to risk analysis related to the transport of hazardous materials by agent-based simulation. He is currently working as an associate professor of computer Engineering at the American University of the Middle East in Kuwait. His research interests include artificial intelligence, multi-agents modelling and simulation, and risk analysis. He can be contacted at email: hassan.kanj@aum.edu.kw.







Nour Mostafa     received the Ph.D. degree from the Queen's University Belfast School of Electronics, Electrical Engineering and Computer Science, UK, in 2013. He was a Software Developer with Liberty Information Technology, UK. He is currently an Associate Professor of computer science with the College of Engineering and Technology, American University of the Middle East. His current research interests include cloud, fog and IoT computing, grid computing, large database management, artificial intelligence, and distributed computing. He can be contacted at email: nour.moustafa@aum.edu.kw.



Zakwan Al-Arnaout     received his Ph.D. degree from the School of Engineering and Computer Science at Victoria University of Wellington, Wellington, New Zealand. He is currently working as associate professor and Department chair of the IST/TNT Department at the American University of the Middle East-Kuwait. His research interests include content caching and replication, multi-hop wireless networks, distributed denial of service attacks detection and prevention, and key management schemes in WSNs. He is a member of the IEEE and the IEEE Computer Society. He can be contacted at email: Zakwan.AIArnaut@aum.edu.kw.



Hassan Harb     received the Ph.D. degree in Computer Science from the Lebanese University (Lebanon) and the University of Franche-Comté (France) in 2016. He held post-doc positions in Ensta-Bretagne, Artois and Paris-Saclay universities between 2018 and 2022. Currently, he is an assistant professor at the American University of the Middle East, Kuwait. He has published more than 80 conference and journal papers and served as a reviewer for a wide variety of IEEE, ACM, Elsevier, Wiley and Springer journals. His research interests are in sensor networks and IoT, artificial intelligence, healthcare, and modular robotic systems. He can be contacted at email: has-san.harb@aum.edu.kw.