Al-Kitab Journal for Pure Sciences (2026); 10(1):179-206.
https://doi.org/10.32441/kjps.10.1.p12

Al-Kitab Journal for Pure Sciences
ISSN: 2617-1260 (print), 2617-8141(online)
https://isnra.net/index.php/kjps

Comprehensive Review of Software-Defined Networking and Application of Machine Learning in Load Balancing

Ahmed Ismael Hadi

Islamic Azad University, Science and Research Branch, Tehran, Iran

*Corresponding Author: ahmadhadi276@gmail.com

Citation: ***Ahmed Ismael Hadi*** . **Comprehensive Review of Software-Defined Networking and Application of Machine Learning in Load Balancing**. Al-Kitab J. Pure Sci. [Internet]. 2025 Jul. 26; 10(1):179-206. DOI: https://doi.org/10.32441/kjps.10.1.p12.

Keywords: Software-Defined Networking, Load Balancing, Machine Learning, Deep Learning.

Article History

Received	16 Jul. 2025
Accepted	26 Jul. 2025
Available online	01 May. 2026

Abstract:

This paper presents a comprehensive review and experimental analysis of machine learning algorithms applied to load balancing in Software-Defined Networking (SDN). From a review standpoint, it considers the rapid development of data centers and the increasing complexity of traffic, underscoring the shortcomings of traditional algorithms. The studies examined are systematically compared in terms of their pros and cons in real-life situations. In addition to the review, the paper presents original findings by assessing specific machine learning techniques, notably Artificial Neural Networks (ANN) and Deep Learning (DL) models, via tenfold cross-validation. The experimental results indicate that the ANN achieves the lowest average response time (1.955 ms), closely followed by the DL (1.962 ms), which demonstrates the highest stability across runs. SVM with C=100C=100C=100 gets the best classification accuracy in ten-fold cross-validation. DL comes in third, and ANN comes in last. When considering both latency and accuracy, DL offers the best overall trade-off between speed, stability, and accuracy for SDN load balancing. It beats traditional baselines (LR: 4.461 ms; SVM: 17.9–18.0 ms). These results, which are directly related to the method used, show that advanced ML methods are better for SDN load balancing. The paper also addresses

significant challenges, such as scalability and adaptability, and proposes future research directions for hybrid AI-driven models to enhance the efficiency of SDN-based network management.

Keywords: Software-Defined Networking, Load Balancing, Machine Learning, Deep Learning.

مراجعة شاملة للشبكات المعرفة بالبرمجيات وتطبيق التعلم الآلي في موازنة الأحمال

أحمد إسماعيل هادي

جامعة آزاد الإسلامية، فرع العلوم والبحوث، طهران، إيران

الخلاصة:

يقدّم هذا البحث مراجعة شاملة وتحليلاً تجريبياً لخوارزميات التعلّم الآلي المطبّقة على موازنة الأحمال في الشبكات المعرفة بالبرمجيات (SDN). ومن منظور المراجعة، يتناول البحث التطور السريع لمراكز البيانات والتعقيد المتزايد لحركة المرور، مبرزاً قصور الخوارزميات التقليدية. وقد جرى إجراء مقارنة منهجية بين الدراسات السابقة من حيث المزايا والعيوب في المواقف الواقعية. إضافةً إلى ذلك، يعرض البحث نتائج أصلية من خلال تقييم تقنيات محددة في التعلّم الآلي، ولا سيما الشبكات العصبية الاصطناعية (ANN) ونماذج التعلّم العميق(DL) ، وذلك باستخدام التحقق المتقاطع ذي عشرة أقسام. وتشير النتائج التجريبية إلى أن ANN حققت أقل متوسط زمن استجابة (١,٩٥٥ ملي ثانية)، تليها مباشرةً DL (١,٩٦٢ ملي ثانية) التي أظهرت أعلى درجات الاستقرار عبر التكرارات. أمّا آلة متجه الدعم (SVM) عند C=100 فقد حققت أفضل دقة تصنيف في التحقق المتقاطع، فيما جاء أداء DL في المرتبة الثالثة و ANNفي المرتبة الأخيرة. وعند النظر إلى كلٍّ من الكمون والدقة، قدّمت DL أفضل توازن إجمالي بين السرعة والاستقرار والدقة في موازنة الأحمال ضمن بيئةSDN ، متفوّقةً على الخطوط الأساسية التقليدية (LR: 4.461 ملي ثانية؛ SVM: 17.9–18.0 ملي ثانية). وتُظهر هذه النتائج، المرتبطة مباشرة بالمنهجية المستخدمة، أنّ أساليب التعلّم الآلي المتقدمة أكثر كفاءة لموازنة الأحمال في SDN. كما يعالج البحث تحديات جوهرية مثل قابلية التوسع وقابلية التكيّف، ويقترح اتجاهات بحثية مستقبلية نحو تطوير نماذج هجينة مدعومة بالذكاء الاصطناعي لتعزيز كفاءة إدارة الشبكات القائمة على SDN.

الكلمات المفتاحية: الشبكات المعرفة بالبرمجيات، موازنة الأحمال، التعلم الآلي، التعلم العميق.

1. Introduction:

The new revolution in networks is the SDN. In traditional networking devices, the control and data plane is integrated. With an increase in the network, the systems become complex and difficult to handle. This can be overcome with the help of SDN.

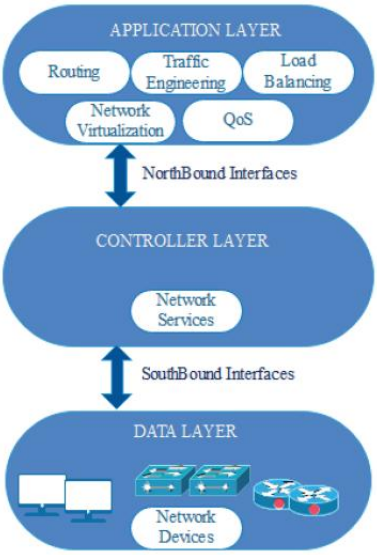

Figure 1: The generic SDN architecture [1].

SDN consists of three layers: data, control, and application. It is highly contrary to traditional network architecture where SDN separates the control plane from the data plane. The separation mainly allows for the management and programming of the network with much ease. The data layer forms the infrastructure comprising switches and routers, while the control layer is seen to be a "brain" since it centrally manages network operations through its interaction with both the data and application layers regarding networking terminology, an "openflow controller" can be thought of as a kind of supervisory application that works inside an SDN environment. The application layer hosts applications that may change the network structure by giving remote access to the controller through northbound APIs, for example, REST. Several southbound protocols such as BGP, NETCONF, XMPP, OVSDB, MPLS-TP, and especially OpenFlow enable communication with the data layer. Among them, the most used southbound protocol is OpenFlow. It allows for more flexible and efficient network management and operation. The major machines will connect to the edge level, while servers are placed at the aggregation and core levels.

• K value in fat tree structure determines how many servers each level will support and the total number of servers that can be connected.

• Decisions are based on the k parameter that helps to make more structured connections. It, therefore makes it rather straightforward to identify the network switch through which the traffic will flow.

Load balancing has two major concepts. First, it is an approach designed to prevent resource overload by shifting a load from a source to multiple sources for sharing in order to help achieve maximum scalability within the network. It is also applied to share one high-load operation among many node devices for parallel processing, which largely boosts up the whole performance. This technique reduces response time, consumption of resources, and load from any single source as well. Most frequently, examples of load balancing include server load balancing and link load balancing. Traditional techniques are based on both hardware and software. In conventional networks, however, there is a difficulty in having an overall view of network status. Thus, it may be difficult to apply the strategy of global load balancing easily. Furthermore, traditional approaches are not able to keep pace with the dynamics taking place concerning the conditions of networks. Another weakness of traditional methods lies in the reliance on specific hardware configurations, which may be expensive, unchangeable, and bound to certain vendors. By contrast, in SDN the load balancer is easily implemented in programmatic code at the controller, so more effective management of network load is achievable [2]. Therefore, two kinds of primary techniques for load balancing are the ones applied in IP networks and those deployed online in the context of an SDN environment. In traditional IP-based load balancing, this process is performed with the support of an LBR. A newly arrived packet first passes through an LBR at the entrance of the network that, after being processed, forwards the packet to one of the available servers based on recent traffic flow conditions by appending the IP address of the server to the packet. The packet is then routed along a computed path to the destination using predefined routing protocols. On the other hand, independently in each and every new packet, SDN performs real-time low-latency server selection and makes immediate decisions on routing with options to send packets across any of the available paths instantaneously [2]. SDN can then classify load balancing techniques as static, dynamic, or a hybrid. In static load balancing, the rules of routing are hardcoded into the load balancer. This might not be effective since it may hurt network performance due to a lack of real-time information about the network. Dynamic schemes, on the other hand, do better in terms of efficiency by distributing loads interactively in accordance with a programmed model; thus, they can adapt to changes in network conditions much more easily. This flexibility in the way traffic is handled makes SDN a fitter platform to implement advanced strategies for load balancing. Among many other fields, machine learning has now emerged as a transformative force on networking. The ability to analyze a huge volume of network data in real time opens the possibility of proactive and effective decision-making capability that becomes quite crucial

in SDN-based systems. These algorithms can predict traffic patterns based on learning from historical data and make decisions that considerably improve the strategy for load balancing. It has also been demonstrated by researchers that ML can improve the efficiency and effectiveness of SDN load balancing in order to handle issues such as traffic congestion and generally enhance network performance [2]. Load balancing is an important aspect of SDN because it ensures resources within the network are used optimally without causing any network bottlenecks or imbalances in traffic. The conventional algorithms in load balancing are based on static configurations that cannot adapt dynamically to the changing conditions of a network. With the integration of ML, the process of load balancing becomes flexible because such algorithms can be dynamically adjusted in accordance with real-time network performance metrics. Indeed, current research depicts that most of the deep learning models achieve substantial enhancements compared to the traditional techniques by reducing response times and improving system accuracy [3]. Despite all the advantages of machine learning integration in SDN load balancing, several challenges are still persisting. However, at this point, the delicacy of models, scalability-related problems, and need for immense computational power to train deep learning models stand in the way of their wide diffusion. Besides, ensuring those models support real-time decision-making without introducing latency into them is still a very active area of research. Several optimization techniques have been studied for reducing the computational overhead of these models for better efficiency in large-scale networks. ML becomes an instrumental factor in enhancing network performance with the evolution of SDN. The future research will be focused on the large-scale and efficiency enhancement of such models to deal with the tough situations occurring in modern DCNs. By integrating the centralized control principle of SDN with predictive capability from machine learning, more robust and adaptive load-balancing solutions can be built that can meet increasing demands from infrastructural underlying networks.

2. Related Work

A. Traditional load balancing techniques in SDN

- ***Conventional Load Balancing Techniques***

Network performance management requires traditional load balancing methods, which utilize algorithms such as round-robin, equal-cost multipath routing, least connections, and random techniques. Every method has its own traits and situations where it works best. We use

different performance metrics to evaluate their effectiveness, as explained in. For instance, a round-robin approach fairly divides requests among servers, which helps prevent starvation but adds some overhead due to queuing [6][8]. Several researchers have thus suggested enhancements to these approaches. One approach is the implementation of load sharing through the duplication of controllers within Virtual SDN controllers [6]. The Virtual SDN architecture allows the master controller periodically to implement virtual IP addresses to other controllers in order to share loads. With failover [8][7] the distributed controllers can further split TCP and UDP traffic to make services even more reliable. The issue of controller placement is also examined in various studies of SDN environments, where the optimal locations are those that minimize latency and ensure seamless operation. The controller placement challenge is also addressed in various studies within SDN environments, where optimal locations minimize latency, ensuring reliability [9] [11]. Hierarchical architectures optimize controller and switch placements to improve load balancing [13]. These methodologies indicate significant improvements in both migration costs and overall performances, especially with increasing traffic [15][12]. Switch migration's role in load balancing has gained attention [6]. BalCon and BalConPlus are two methods that reduce migration costs while ensuring an even load distribution [14]. The Flow Stealer method is a lightweight approach that allows idle controllers to share workloads for a short time [16]. These kinds of new ideas make it easier to respond to changes in traffic in dynamic settings [18]. Adding energy-efficient technologies to load-balancing strategies is a significant step forward [19][20]. Using green energy helps make the best use of resources and reduces the need for traditional power sources. Advanced resource allocation processes are made possible by algorithms like QALB and hybrid flow-based methods [21][22]. In general, traditional load balancing methods are continually evolving to address new challenges and improve network performance [23][24].

- ***Artificial Intelligence Based Load Balancing Techniques***

In these regards, artificial intelligence techniques utilize metaheuristics approaches in efficiently solving real-world problems of load balancing issues in SDN. AI includes several sub-classes, namely deep learning, neural networks, natural language processing, and decision-making strategies. Characterizing different AI-based techniques for Load Balancing is discussed along with a discussion on performance metrics utilized for evaluating such techniques. These AI techniques and methods will improve learning and decision-making in SDN and will thus facilitate effective load distribution across network resources. Researchers

have also developed heuristic methods to move switches from overloaded controllers to under loaded controllers. For instance, the heuristic in [24] incorporates movements and shifts within a search framework that assesses the potential of every step in migration. This method does not stop working when it detects that a simple migration is impracticable; instead, it investigates complex movements to achieve an optimal performance. Moreover, authors in [25] proposed a publish/subscribe system based on SDN that disseminates efficiently events using a global topology view and Huffman trees to minimize end-to-end latency. Other key techniques of optimization are essential to balance the loads and reduce congestion during different phases. In the work presented by [26], the first two phases create a sub-topology for performance enhancement, which is followed by load balancing. The authors in [27] developed the OCLB method, which would reduce the average response time of controllers using pre-calculated switch migrations based on real-time application data. Moreover, genetic optimization techniques, as shown in [28], can be used to minimize load imbalance in elastic optical networks using a phased approach that guarantees optimal solutions. A variety of techniques improves load-balancing strategies using AI methodologies like PSO and fuzzy logic. For example, the modified PSO technique in [29] efficiently selects the vehicular nodes for reducing power consumption whereas KKT conditions used in [30] help to select the appropriate controller during load balancing. The fuzzy logic in [31] analyzes server load parameters due to which dynamic adjustment of server performance is done to optimize the energy consumption and achieve load balance effectively. Lastly, game theory has been used in traffic balancing among the SDN controllers as obtained in [32]. These methods therefore train the networks to learn about the right flow rates to attain an equilibrium state that enhances load distribution. A greedy approach is employed in [33] for efficient switch migration with the aim of effectively balancing the load, considering migration costs and load variations. Moreover, the framework in [34] investigates multi-criteria decision-making for choosing the target controllers, thereby optimizing the process of load balancing in the environment of SDN.

Table 1: Comparative Summary of Traditional and AI-Based Load Balancing Techniques in SDN

Approach Type	Algorithms / Techniques	Characteristics	Strengths	Limitations	Common Evaluation Metrics
Traditional (Conventional)	Round Robin, ECMP, Least	Static rules based on topology or connection count	Simple to implement, low	Lack of adaptability to dynamic conditions, limited	Response time, end-to-end latency, resource

Approach Type	Algorithms / Techniques	Characteristics	Strengths	Limitations	Common Evaluation Metrics
	Connections, Random, LBR		computational overhead	global view, dependency on specific hardware	utilization, switch migration cost
Enhanced Traditional	BalCon, BalConPlus, Flow Stealer, Controller Placement Optimization	Dynamic migration or multi-controller deployment	Reduced migration cost, improved reliability	Still limited in traffic prediction and adaptability	Controller delay, scalability, service reliability
AI-Based	ANN, Deep Learning, SVM, PSO, Fuzzy Logic, Game Theory, Genetic Algorithms	Learning-based and metaheuristic models	Dynamic adaptation to network changes, traffic prediction, multi-criteria optimization	High computational demand, model complexity, possible added inference latency	Classification accuracy, response time, throughput, energy efficiency
Hybrid Models	Hybrid ML + Heuristics (e.g., OCLB, QoS-aware ML)	Integration of ML with heuristic/decision-making methods	Balance between high accuracy and low response time	Complex design and training, large data requirements	Scalability, flexibility, congestion avoidance success rate

B. MACHINE LEARNING IN SDN

SDN has a centralized controller that can see an overall picture of the network and hence allows for easier management and control. Machine learning techniques will further enhance these capabilities through advanced data analysis, optimization of the network, and automatic provisioning of services. This will, in turn, enable the controller to self-learn and optimally make decisions in response to changes in the network on its own. This section highlights various challenges that require the use of machine learning in overcoming many challenges related to SDN: classification of traffic, routing optimization, prediction of QoS and QoE, managing resources, and security. Further, we describe how machine-learning algorithms are put into place in the context of the SDN.

1) Elephant Flow-Aware Traffic Classification

Elephant flow-aware classification is used to classify two kinds of flows: elephant flows, which are long-lived and bandwidth-intensive, and mice flows, which are short-lived and delay-

sensitive. In data centers, roughly 80% of flows are mice flows, while most of the volume of data is carried by elephant flows. Classifying elephant flows is thus of vital importance for efficient traffic management. According to various researches, it is possible to allow the central SDN controller to apply other more effective traffic optimization techniques according to the classification result using machine learning techniques for classifying traffic at the network edge. One argued that within SDN, detection of elephant flows could be done by cost-sensitive learning. The approach comes with a two-phase strategy wherein it first measures the head packet in order to distinguish suspicious elephant flow from mice flow. Afterwards, in its second phase, it uses a decision tree to verify the flows if they are truly elephant or not. The proposed elephant flow detection system consists of three major components: Flow Collector, Real-time Detection Strategy, and Classifier. The Flow Collector is responsible for collecting the input to the system by capturing the IP packets from the SDN network, and the flows will be recognized depending on IP packet header inspection. Each flow is identified by using a five-tuple that repre-sents source and destination information and statistical features of duration and packet count. In the processing stage, the Real-time Detection Strategy module will extract key features from the collected flows to classify them effectively. This module shall adopt a hybrid strategy that encompasses TCP flow characteristics and DPI attributes to select the relevant n-tuple features, essential in the correct identification of elephant flow. The objective of this module shall be to have an appropriate feature selection to develop an efficient and robust classification. The Classifier module classifies each n-tuple precisely into an elephant flow or not following the feature extraction. It may employ any statistical method or machine learning algorithm to do the classification; in **[36]**, the C4.5 decision tree is used as the detection method. Another contribution proposed here is the cost-sensitive approach in order to improve the efficiency of the elephant flow detection, since accurate elephant flow detection can help with optimal resource allocation within the network and avoid congestion. This system enables the correct identification of elephant flows by users and the implementation of necessary optimizations in traffic management. Fig. 2.

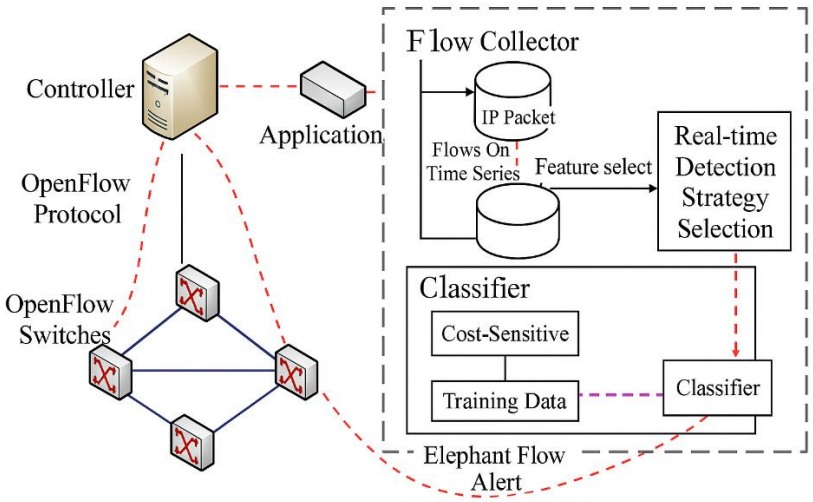

Figure 2: The System model detecting elephant flows[36].

In this section, we present a two-stage strategy for elephant flow detection, as illustrated in Fig. 2. This method is designed to quickly and efficiently identify elephant flows using statistical thresholds and flow features over time series.

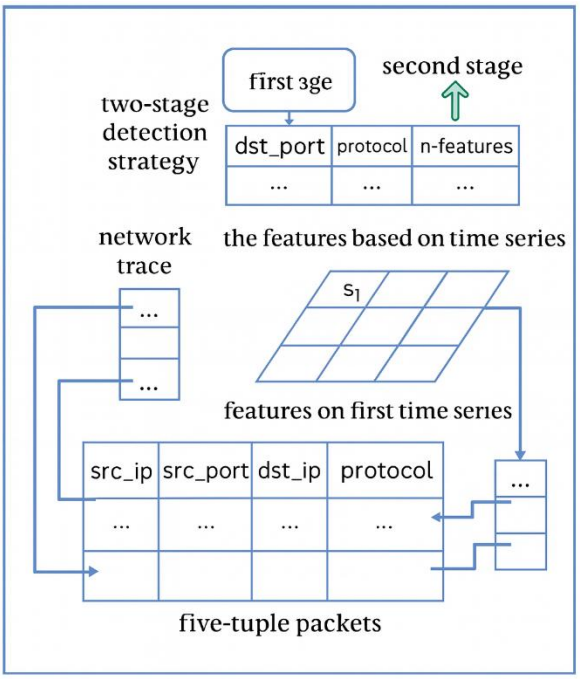

Figure 3: Two-Stage Elephant Flow Detection Strategy [36].

In the first stage, we distinguish large flows from small flows using head packet measurement. This differentiation is based on thresholds such as destination port and protocol.

The suspicious flows identified at this stage are forwarded to the second stage to enhance accuracy. In the second stage, n-features are extracted from the statistical data collected in the initial time series, including parameters like c2s_pkts and s2c_psmax. To optimize the feature set for the dataset, we employ a correlation-based filter (CFS), which plays a crucial role in the feature selection process [36]. This two-stage approach enables more precise and efficient detection of elephant flows, which is particularly significant for improving traffic management in SDN networks.

2) Application-Aware Traffic Classification

Machine learning methods for identifying internet traffic types typically work with data collected from specific protocols known as IPFIX. These protocols assist Internet Service Providers (ISPs) in gathering and processing traffic data from routers. IPFIX aggregates packets to provide information about flows, such as duration and bit rate, which are then used in a separate system for traffic classification [37]. Software-Defined Networks (SDN) offer new approaches for collecting and managing this data. OpenFlow, a protocol for SDN, directly gathers flow statistics and connects packets to flows based on specific rules. This allows information to be collected from various points in the network, ranging from low-traffic access switches to high-traffic core switches. One of the advantages of SDN is that statistics and information are sent directly to the controller, enabling it to respond quickly and automatically. For example, if the controller learns information about a traffic flow, it can swiftly apply new rules to manage that flow. Additionally, SDN allows for customization of the types and numbers of packets that need to be analyzed. In general, most of the procedures of traffic classification in SDN consist of several major steps. An SDN application connects to switches using the OpenFlow protocol and instructs them to forward all incoming traffic to the controller. These include recorded TCP packets through port mirroring or direct data collection, respectively, capturing important information such as transport protocol and flags of the TCP. It inspects the packets, identifies, counts, and logs the first five packets exchanged between the client and the server. After that, all the remaining packets are handled locally inside the switch, in order to minimize the load of Packet In messages at the controller (Fig 3). After data collection and analysis, it creates a labeled dataset to train machine learning models. Therefore, the data is divided into a training-testing set, and models trained on algorithms such as Random Forest and Gradient Boosting are used. For the measurement of the accuracy of these models, the results predicted by them are compared against the actual labels. Further, the average of these is

determined. The obtained results can finally be used for flow optimization and managing traffic in SDN networks, which can be further extended for legacy networks [37].

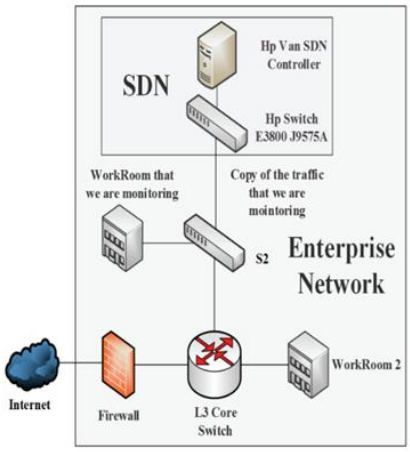

Figure 4: Enterprise Network Setup for Application-Aware Traffic Classification in SDN [37].

Application-aware traffic classification focuses on determining the specific applications associated with various traffic flows. Amaral et al. conducted research in enterprise networks to explore this concept. In their approach, a straightforward OpenFlow-based SDN system was established within an enterprise environment to collect traffic data. Various classification algorithms were employed to categorize the traffic flows based on the identified applications A system known as MultiClassifier was introduced, which integrates both machine learning (ML) and Deep Packet Inspection (DPI) techniques to identify applications. When a new flow is detected, the ML classifier is initially utilized for classification. If the reliability of the ML classifier's output exceeds a predetermined threshold, this result is accepted as the final output of the MultiClassifier. Instead, the DPI method is employed. If the DPI classifier does not return a "UNKNOWN" result, its output is chosen as the final classification. In a related area, another study evaluated the process to classify applications that implement the UDP protocol. Specifically, it proposed a behavioral classification engine to improve the accuracy of application-aware traffic classification. Building upon this, we deployed the Support Vector Machine (SVM) algorithm to categorize UDP traffic based on NetFlow records, which include the number of packets and the total number of bytes. Simulation results indicated that this SVM-based classification engine achieved an accuracy rate of over 90%. Furthermore, research on mobile application classification introduced a framework called Atlas, which aims to identify mobile applications effectively. This framework uses a crowdsourcing approach to gather accurate data from end devices, which is then utilized to train a decision tree. The resulting

model is capable of identifying mobile applications associated with the analyzed traffic flows. Simulation outcomes demonstrated that Atlas achieved an average classification accuracy greater than 94% for the top 40 applications available on Google Play. In another research effort, deep neural networks (DNNs) were utilized to identify mobile applications. Data was collected from an experimental network regarding mobile network traffic, and five flow features destination address, destination port, protocol type, TTL, and packet size were selected to train an 8-layer DNN model. The simulation results revealed that this model attained a remarkable accuracy of 93.5% in recognizing 200 distinct mobile applications. Mobile applications often produce various types of traffic flows. For example, the Facebook application can generate video, voice, instant messaging (IM), and file sharing traffic flows. To facilitate a detailed and fine-grained classification of mobile application-aware traffic, it is crucial to identify both the specific mobile applications and the corresponding types of flows. A sophisticated system called TrafficVision was proposed for this purpose in an SDN-enabled wireless edge network. The TrafficVision Engine (TV Engine) is the most important part of TrafficVision. It is a crucial component of the overall design and operation of the SDN-enabled system. Figure 2 illustrates the architecture of the TrafficVision Engine (TV Engine), comprising three main components. The first thing it does is collect, store, and analyze flow statistics and ground-truth training data from end-user devices and access points. This basic step is crucial for ensuring that applications are correctly identified. Second, the TV Engine utilizes a decision tree classifier to identify various application names, such as Facebook, YouTube, and Amazon. It also utilizes a k-Nearest Neighbors (k-NN) classifier to distinguish between different types of flows, such as video content, audio files, and video chats. Together, these components facilitate a detailed and efficient mobile application-aware traffic classification system within SDN-enabled wireless edge networks.

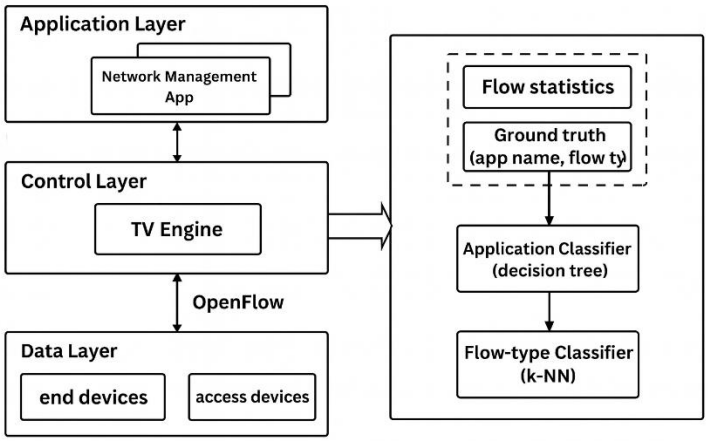

Figure 5: The architecture of Traffic Vision, which is designed for Software-Defined Networking (SDN), operates at a high level and includes the workflow of the TV Engine

The TV Engine performs three primary functions: (1) it gathers flow statistics and authentic training data from both end devices and access points, (2) it utilizes a decision tree classifier to determine the names of applications, and (3) it employs a k-nearest neighbors (k-NN) classifier to categorize different types of data flows.

3) Impact of QoS-Aware Traffic Classification on SDN Performance

QoS-aware traffic classification has a high impact on the performance of Software-Defined Networks. In this method, an SDN can manage flows according to their QoS class requirement, specified in terms of maximum delay, jitter, and packet loss rate. This way, network resources are utilized much more intelligently. Delay-sensitive traffic-that is, video calls or other similar real-time applications-will be processed quicker. Therefore, the network will be able to produce higher performance and yield better user experience in respect to various applications. It is hard to identify all the applications in the internet environment; QoS classification classifies the traffic flows according to QoS requirements rather than identifying each application. This enables SDNs to manage traffic in a more effective way without requiring precise knowledge of specific applications. Using semi-supervised learning and some techniques like Deep Packet Inspection, the network can have an exact classification and management even in unknown traffic flows, improving the accuracy of the classification and enhancing traffic management. Besides, the QoS-based classification helps SDNs in optimizing the utilization of network resources and forwarding the traffic to meet its actual needs, thereby avoiding congestion and improving overall network efficiency. By applying machine learning

and deep learning algorithms, SDNs manage to process large and complex datasets of flow traffic more efficiently to let the network become more intelligent towards dynamic and changing conditions. In summary, this approach makes for a very important role in augmenting the quality of the services and overall efficiency of the SDN networks. In [35], a QoS-aware traffic classification system is proposed which combines semi-supervised learning algorithms and DPI. Here, the DPI only labels some of the traffic flows from known applications, and then semisupervised learning algorithms, such as Laplacian SVM, use the labeled training data to classify the traffic flows from unknown applications. This provides the capability for the system to efficiently classify both known and unknown traffic into different QoS classes. Simulation results demonstrate that the achieved high accuracy of the developed traffic classification system attains more than 90% in the exactness of the classification for different kinds of traffic flows. This becomes very important for SDN systems because this allows the network to make more realistic decisions about each flow QoS requirements by optimizing resource allocation and traffic management. It not only enhances the precision and efficiency in traffic management but also tackles the challenges brought about by exponential growth in diversified internet applications. Due to the fact that it categorizes the traffic based on QoS needs rather than identifying each application, it proves highly effective in dynamic and variable environments such as SDNs, which may improve performance and reduce congestion in networks.

Table 2: Machine Learning-Based Traffic Classification Solutions in Software-Defined Networks (SDN)

ref	Objective	Learning Model	Dataset Input	Dataset Output
[36]	Elephant flow-aware classification	Decision tree	Basic five-tuple, statistical flow features	Elephant flow, mice flow
[37]	Application-aware classification	Random forest	12 flow features (packet size, inter-arrival time, flow duration, etc.)	8 applications (e.g., YouTube, Skype, Dropbox)
[38]	Application-aware classification	ML classifier	Not mentioned	Not mentioned
[39]	Application-aware classification	SVM	Basic five-tuple, Netflow records (packet, byte counts)	8 applications (e.g., PPstream, Skype, DNS)
[40]	Application-aware classification	Decision tree	Flow features (packet sizes, source/destination ports)	Top 40 apps in Google Play Store
[41]	Application-aware classification	Deep NN	Destination address, protocol type, TTL, packet size	200 mobile applications
[42]	Application-aware classification	Decision tree, k-NN	Flow features (per reference [7])	Dataset 1: 37 apps, Dataset 2: 45 apps
[35]	QoS-aware classification	Semi-supervised learning	9 flow features (packet inter-arrival time, Hurst parameter, etc.)	Four QoS classes (voice/video, streaming, etc.)

3. Machine Learning Techniques for Load Balancing

Machine learning has been realized as one of the innovative ways of load balancing in SDN. It involves the use of mathematical and statistical algorithms to analyze information and predict network behavior. Machine learning will enable the system to recognize the patterns in the traffic and select the best paths which data shall take based on pattern recognition. Such algorithms predict traffic behaviors by analyzing historical and current network data, thus helping network managers make better decisions regarding network traffic. Certain examples of machine learning applications in SDN involve identifying the pattern of traffic load and predicting conditions in the future. For instance, using traffic data gathered, it can be predicted using machine learning algorithms which links and nodes will be highly loaded in the future. Therefore, the SDN controller may make better decisions on the distribution of the traffic over the network and choose routes in order to avoid highly congested paths for routing data. Hence, the efficiency of the network improves and the possibility of bottlenecks and unexpected outages becomes low [43], In addition, machine learning further enhances the response to failures by enabling quick detection of link or node failures through highlighting appropriate alternative paths with which data transmission can continue. In that way, the network remains functional without degradation in performance. Overall, the application of machine learning in the traffic management of SDN networks contributes to load balancing optimization, greatly improving the reliability and efficiency of the network.

C. Supervised Learning Methods

The controller can identify all paths between two nodes and determine the optimal load conditions for each path by updating the network's topology information. This broad view enables the controller to make more informed decisions about managing traffic and balancing loads. Initially, network simulation is conducted to gather data, and then it is repeated to validate the results. The first step in this process is to start the Floodlight controller, which serves as the foundation for the network simulation. After that, Mininet is used to set up the network topology.

At this stage, the user inputs the source host, and then the least loaded host is selected as the destination. Traffic is generated between these hosts, and the Wireshark tool is utilized for monitoring this traffic [43], Next, Dijkstra's algorithm is employed to determine the shortest path(s) between the source and destination. The machine-learning module is enabled to select

the optimal path when multiple shortest paths are available. If not, the shortest path found is the best one. This process helps improve traffic flow and ensures that packets reach their destination as quickly as possible. To manage failures, the SDN controller first identifies link failures, particularly when a switch is unable to send a packet-in message to the controller within a predefined time period. In such cases, the SDN controller notifies the load balancer about the link failure, temporarily removing all paths that include the faulty link and identifying the least loaded path. Additionally, in the event of a node failure, the controller identifies the backup path and uses it to maintain connections between the source and destination hosts. These mechanisms enhance the efficiency and stability of the network while enabling effective load management. The flow diagram of the system is shown in Figure 6.

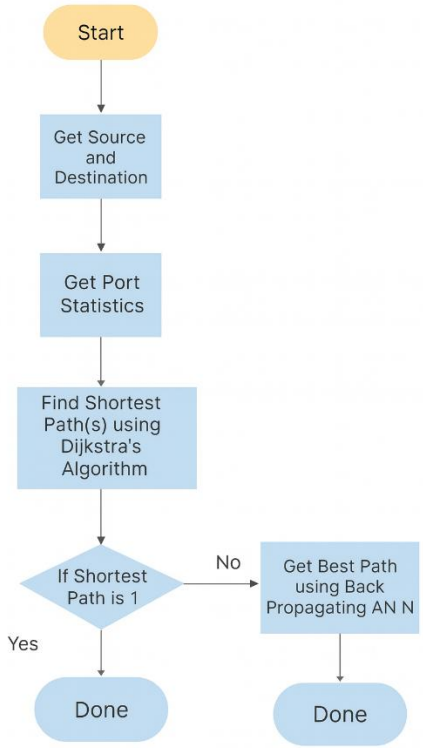

Figure 6: The Machine Learning Techniques for Load Balancing [43]

D. Deep learning models for load prediction and optimization

Deep learning is a way of coming up with machine learning in cascading layers for feature extraction from the input data towards making decisions. What leads the most among principles behind deep learning is that there is learning of data representations at gradually higher levels of abstraction. For almost all levels, higher-level abstract representations get derived from lower-level representations. This is what generally makes this hierarchical process of learning

effective. It can perceive, understand and learn complex representations directly from raw data, which finds an application in many fields [1].

Three or four elements or parts commonly show up in an application:

1) Representation, i.e. how the state of the environment can be represented in handy numerical formats serving as the input layer for the DL network.
2) How the results of the validation shall be represented or interpreted in such a way as to reflect the physical meaning of the output layer of the DL network.
3) How the reward value shall be calculated and updated, together with the determination of an appropriate reward function that will guide the iterative updates of weights in each neural layer.
4) DL architecture, which includes the number of hidden layers, material design for each layer, and layer inter-relations, is done. Deep learning has solved various open issues in many application areas including computer vision, mobile and wireless networks, natural language processing.

Model architecture: The flow chart of the experiment model architecture is depicted in Figure 7. First of all, the Mininet emulator is selected and then a k-ary fat-tree topology is created.

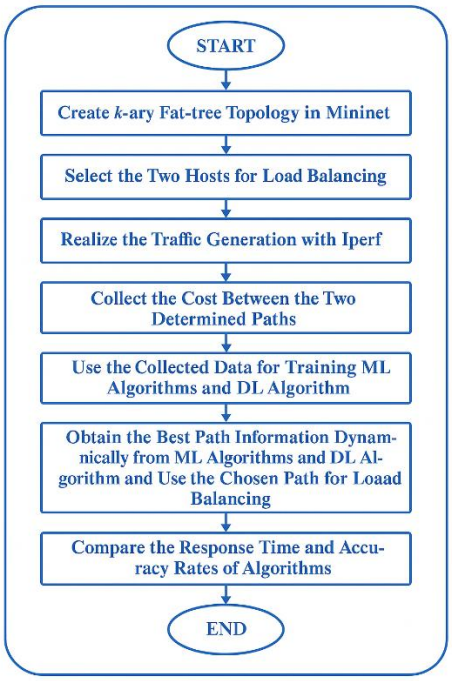

Figure 7: The Deep Learning Techniques for Load Balancing [2].

Then, two hosts are randomly chosen to initiate the load balancing process. For this, tx and rx transmission and receiving rates are enabled in Floodlight. Traffic between them is generated using Iperf. Finally, respective costs between the selected paths are gathered. Route information like IP addresses, switches, MAC addresses, and port mappings are accessible by using Rest-API. This data provides the cost between these two paths in milliseconds on a text file (.txt). This additional file can be retrieved in Ulu. On the Floodlight side, the costs are gathered, and configurations are modified for the enablement of Rest-API operations. The paths cost computation:

(1)

$$Cost = tx(transmitted) + rx(received)$$

Equation (1), adapted from [2], defines the cost as the sum of transmitted (tx) and received (rx) packets. This formulation has been widely used in SDN-related studies as a straightforward performance indicator, and in this work, we adopt it to ensure consistency with prior cost-based approaches. Total cost is calculated as the summation of all tx and rx values. The data so collected forms the training input for ML and DL. In-Port, Out-Port, Source IP, Destination IP, Source MAC, Destination MAC, etc. The program updates this information per minute, thus making it dynamic; once the best path through which data is to be sent is estimated using algorithms through ML and DL, then this information is used for load balancing. The block diagram of the load balancing mechanism used in the simulations is illustrated in Figure8.

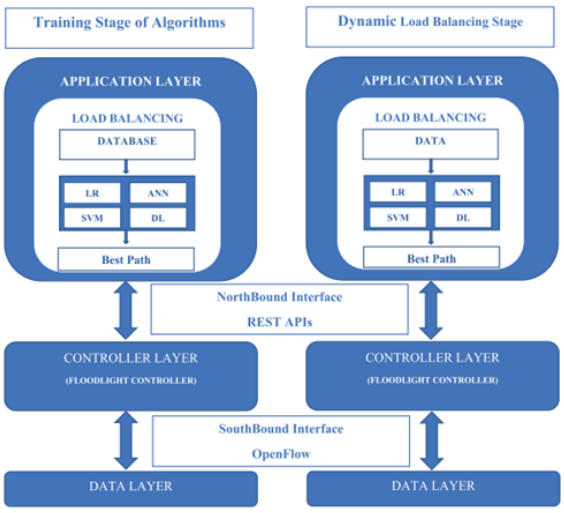

Figure 8: The load balancing mechanism in the simulations [2].

Two situations are looked at. In the first case, connections are made in MiniEdit, and two hosts (Host 1 [h1] and Host 4 [h4]) are chosen to share the load. The REST API gathers information about the costs of routes between these two hosts, which helps determine a route that incurs the least cost. The selected paths are h1, switch7 (s7)-switch4 (s4)-switch8 (s8) and h4, h1-s7-switch3 (s3)-s8-h4. Cost values for the paths are obtained using Rest-API by providing the MAC addresses of the switches. An example of a data entry might look like "′ 02 :: 15 :: 01′ : 288, ′ 02 :: 0a :: 01′ : 1,530." Based on the obtained route information and cost values, classification values are added to the database. A sample data row in the database is "288, 1,530, 0." In the second scenario, load balancing occurs over different routes than in the first scenarioConnections to MiniEdit are reestablished, and two new routes (h1–h3) and (h6–h7) are selected simultaneously. The REST API collects road cost information for these hosts, which helps them choose the most cost-effective routes. As in the first scenario, classification values are added to the database, with a sample data line looking like "1,254, 694, 1, and 81,659, 80,769, 1." A total of 800 training data points and 192 test data. points are generated. This database is then utilized to train ML and DL algorithms. For ML, algorithms like Logistic Regression (LR), Support Vector Machine (SVM), Artificial Neural Network (ANN), and DL are used to compare results during training and testing. Specifically, the SVM algorithm has a C parameter that can be set to 1 or 100. Regarding the ANN model, as shown in Figure 6, it has two layers: the first layer has five nodes, and the second layer has one node. The learning rate is set to 0.00025. The first layer employs the ReLU activation function, while the second layer utilizes the sigmoid activation function. The paths' input variables are shown as x0 and x1. In the case of DL, Figure 9 shows the four-layer model used.

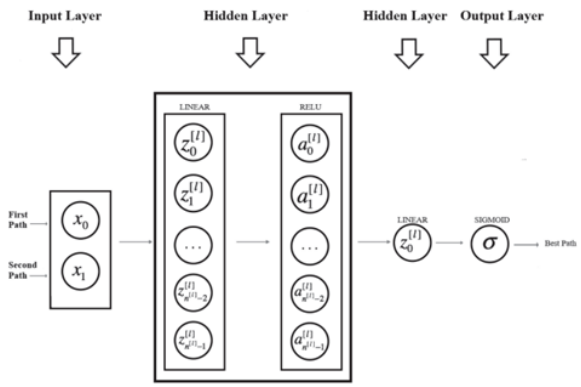

Figure 9: The DL model for load balancing [2].

This model has 20 nodes in the first layer, 7 nodes in the second layer, 5 nodes in the third layer, and 1 node in the fourth layer. The activation function for all layers except the fourth is the ReLU function. The activation function for the fourth layer is the sigmoid function. The input variables for the paths are still x0 and x1, and the notation **[l]** shows which layer they belong to. The letter Z stands for the pre-activation parameter.

4. Experimental Evaluation

In their study, B. Babayiğit and B. Ulu used deep learning techniques to optimize load balancing in SDN within data centers. They tried several machine learning models, including deep learning techniques that improve the network performance-traffic management. The research study in the paper gives insight into how AI-based methodologies can enhance both scalability and responsiveness of SDN-based networks and, finally, provide an efficient way of solving load balancing strategies **[2]**. These measurements are response times in milliseconds and are shown in Figure 10 and listed in Table 3. The researchers tested the LR, SVM, with C = 1 and C = 100, ANN, and DL algorithms in their experiments. Therefore, these measurements give a view of each algorithm's performance under several different network conditions. The DL model for load balancing **[2]**.

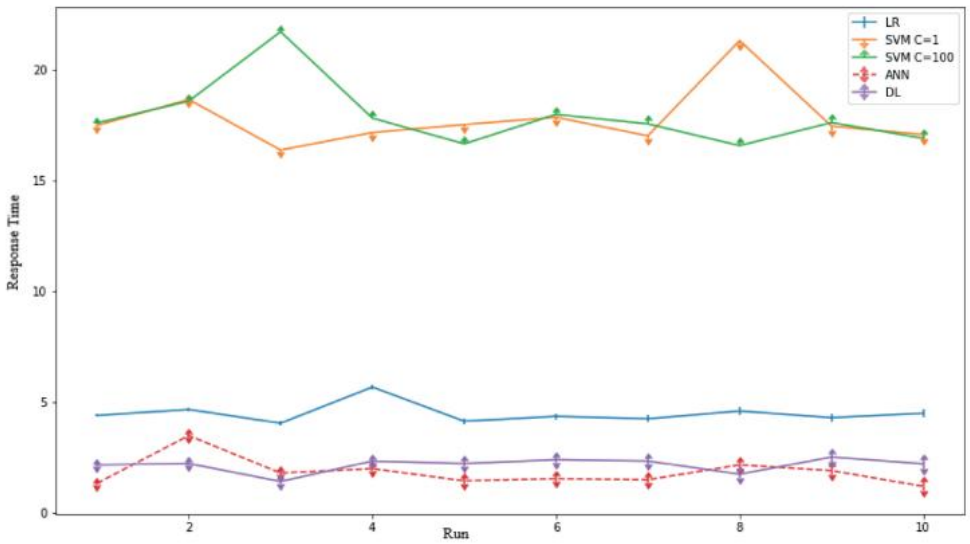

Figure 10: The response times for the LR, SVM (with C = 1 and C = 100), ANN, and DL algorithms in the second scenario [2].

Based on Figure 10, a clear performance gap is observed among the evaluated algorithms in terms of response time across ten runs. The SVM classifiers (C=1 and C=100) exhibit the highest response times, ranging from approximately 16 to 22 milliseconds, and show noticeable fluctuations, indicating lower stability. Logistic Regression (LR) gets average results, with response times of about 4–5 milliseconds, and its performance is fairly stable. The Artificial Neural Network (ANN), on the other hand, typically has the fastest response times, which are usually between 1.5 and 2.5 milliseconds, although they vary slightly from run to run. Deep Learning (DL) models work almost as well as ANN, but they are more stable and don't change as much. Their response times are a little longer, ranging from 1.5 to 2.7 milliseconds. In general, ANN is the fastest algorithm, and DL is the most reliable and consistent. This shows that advanced ML techniques are better than older ones like SVM and LR for SDN load balancing.

Running	LR	SVM C = 1	SVM C = 100	ANN	DL
1	4.20	17.32	17.79	2.10	1.97
2	4.97	18.17	18.02	1.32	1.57
3	4.09	16.53	20.98	1.82	1.63
4	4.94	17.92	18.65	1.29	1.27
5	5.11	17.46	17.08	1.94	2.37
6	4.77	18.10	18.01	2.68	2.04
7	4.03	19.40	16.56	1.68	2.70
8	4.10	20.12	16.72	2.64	1.57
9	4.25	18.96	18.12	2.44	1.75
10	4.15	16.09	17.03	1.64	2.75

Table 3: The response times (in milliseconds) for the LR, SVM (with C = 1 and C = 100), ANN, and DL algorithms in determining the optimal path for the second scenario [2].

This table presents a comparative analysis of Logistic Regression (LR), Support Vector Machine (SVM) with C=1 and C=100, Artificial Neural Networks (ANN), and Deep Learning (DL) in ten consecutive runs in terms of response time (milliseconds). The results reveal a significant performance gap among the evaluated methods. The SVM algorithms, under both C values, demonstrated the weakest performance; their response times ranged between 16 and 22 milliseconds, and in addition to having a high average, they showed considerable fluctuations across runs, indicating low stability. In contrast, LR showed moderate performance with response times in the range of 4–5 milliseconds. While slower than ANN and DL, LR maintained more consistent results compared to SVM.

In contrast, ANN and DL outperformed other methods. ANN consistently had the fastest response times, ranging from 1.3 to 2.6 milliseconds, resulting in the lowest average delay. DL performed nearly as well, but was more stable with less variation, posting response times from

1.2 to 2.7 milliseconds. These small differences suggest ANN is ideal for applications needing low latency, while DL is preferable for environments that value stability and adaptability. Overall, the table demonstrates that although traditional approaches such as LR and especially SVM are computationally simpler, they cannot compete with ANN and DL in terms of response time for SDN load balancing.

5. Challenges and Future Directions

Along with continuous growth in the area of machine learning and load balancing in SDN, a number of works remain to be pursued towards surmounting challenges. Among those are the responsiveness and adaptability enhanced by algorithms under dynamic environments. Network conditions fluctuate due to rapid changes in traffic loads, failures of nodes, and modification within network topologies. An algorithm showing quick adaptability with optimal performance against such changes has their importance in such scenarios. Another challenge is scalability: the network sizes are increasing each day, which increases the computational complexity of the load balancing algorithms, hence worst response time and less efficiency. These need scalable solutions that can ensure better performance on larger networks. Also, there is a need for more scope in integrating AI techniques within the existing architecture of SDNs. This integration can make different decisions for various processes in real time to help enhance the overall efficiency in load balancing mechanisms. Future directions may relate to exploring hybrid models leveraging the strengths of different machine learning techniques to exhibit superior performance. There is also the security perspective, which always remains a very critical concern. Since load balancing algorithms are achieved by the sharing of data and communications between nodes, the confidentiality and integrity of their data are crucial. It is highly recommended that studies on secure techniques for load balancing against eventual vulnerabilities be carried out in order to increase reliability within SDN environments in general.

6. Conclusion

Machine learning algorithms for load balancing in SDN show promise as a way to make networks more efficient and better manage resources. Although ANN and DL models have demonstrated better response times and accuracy, there are still some issues that need to be addressed before they can be applied in real-life scenarios. One significant issue is that deep

learning models are computationally expensive to run, requiring substantial processing power, memory, and time to train. In large-scale SDN deployments, these requirements may lead to scalability bottlenecks and increased operational costs unless optimized through techniques such as model pruning, quantization, or GPU/TPU acceleration.

Another challenge is the need for models to adapt to evolving traffic patterns. Recently, hybrid strategies that integrate traditional optimization methods (such as Dijkstra's algorithm and heuristics) with AI-driven models have demonstrated effectiveness. Combining static, rule-based techniques with deep reinforcement learning can achieve a balance between real-time adaptability and reduced computational load. Ensemble approaches that integrate ANN, SVM, and decision trees can further bolster system resilience to diverse traffic types.

Future research should focus on developing resource-aware AI models that can dynamically scale according to available computing resources while maintaining accuracy. Also, energy-efficient versions of DL models are needed to make deployment possible in real-world data center networks. These models could use edge computing or lightweight federated learning methods. To ensure that SDN load-balancing systems are more accurate, responsive, cost-effective, sustainable, and scalable, these problems must be addressed.

7. References

[1] Tennakoon, Deepal, Suneth Karunarathna, and Brian Udugama. "Q-learning approach for load-balancing in software defined networks." 2018 Moratuwa engineering research conference (MERCon). IEEE, 2018.

[2] B. Babayiğit and B. Ulu, "Deep learning for load balancing of SDN-based data center networks," International Journal of Communication Systems, vol. 34, no. 6, 2021. DOI: 10.1002/dac.4760.

[3] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015. DOI: 10.1109/JPROC.2014.2371999.

[4] M. Alenezi, M. K. Jha, and R. Shrestha, "A machine learning approach for load balancing in SDN: Current trends and challenges," IEEE Access, vol. 9, pp. 31815-31827, 2021. DOI: 10.1109/ACCESS.2021.3061122.

[5] G. Wang, T. S. Eugene Ng, and A. Shaikh, "Programming your network at run-time for big data applications," ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 67-74, 2014. DOI: 10.1145/2656877.2656888.

[6] S. Ejaz, Z. Iqbal, P. A. Shah, B. H. Bukhari, A. Ali and F. Aadil, "Traffic load balancing using software defined networking (SDN) controller as virtualized network function", IEEE Access, vol. 7, pp. 46646-46658, 2019.

[7] H. Gasmelseed and R. Ramar, "Traffic pattern-based load-balancing algorithm in software-defined network using distributed controllers", Int. J. Commun. Syst., vol. 32, no. 17, pp. e3841, Nov. 2019.

[8] H. Jin, G. Yang, B.-Y. Yu and C. Yoo, "TALON: Tenant throughput allocation through traffic load-balancing in virtualized software-defined networks", Proc. Int. Conf. Inf. Netw. (ICOIN), pp. 233-238, Jan. 2019.

[9] P. Tao, C. Ying, Z. Sun, S. Tan, P. Wang and Z. Sun, "The controller placement of software-defined networks based on minimum delay and load balancing", Proc. IEEE 16th Intl Conf Dependable Autonomic Secure Comput. 16th Intl Conf Pervas. Intell. Comput. 4th Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr.(DASC/PiCom/DataCom/CyberSciTech), pp. 310-313, Aug. 2018.

[10] Y. Hu, T. Luo, W. Wang and C. Deng, "On the load balanced controller placement problem in software defined networks", Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC), pp. 2430-2434, Oct. 2016.

[11] Y. Zhao, C. Liu, H. Wang, X. Fu, Q. Shao and J. Zhang, "Load balancing-based multi-controller coordinated deployment strategy in software defined optical networks", Opt. Fiber Technol., vol. 46, pp. 198-204, Dec. 2018.

[12] S. Jiugen, Z. Wei, J. Kunying and X. Ying, "Multi-controller deployment algorithm based on load balance in software defined network", J. Electron. Inf. Technol., vol. 40, no. 2, pp. 455-461, 2018.

[13] Y.-W. Ma, J.-L. Chen, Y.-H. Tsai, K.-H. Cheng and W.-C. Hung, "Load-balancing multiple controllers mechanism for software-defined networking", Wireless Pers. Commun., vol. 94, no. 4, pp. 3549-3574, Jun. 2017.

[14] Y. Xu, M. Cello, I.-C. Wang, A. Walid, G. Wilfong, C. H.-P. Wen, et al., "Dynamic switch migration in distributed software-defined networks to achieve controller load balance", IEEE J. Sel. Areas Commun., vol. 37, no. 3, pp. 515-529, Mar. 2019.

[15] P. Song, Y. Liu, T. Liu and D. Qian, "Flow stealer: Lightweight load balancing by stealing flows in distributed SDN controllers", Sci. China Inf. Sci., vol. 60, no. 3, 2017.

[16] T. Hu, J. Lan, J. Zhang and W. Zhao, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking", Peer-to-Peer Netw. Appl., vol. 12, no. 2, pp. 452-464, 2019.

[17] H. Zhong, Y. Fang and J. Cui, "Reprint of 'LBBSRT: An efficient SDN load balancing scheme based on server response time", Future Gener. Comput. Syst., vol. 80, pp. 409-416, Mar. 2018.

[18] T. Han and N. Ansari, "A traffic load balancing framework for software-defined radio access networks powered by hybrid energy sources", IEEE/ACM Trans. Netw., vol. 24, no. 2, pp. 1038-1051, Apr. 2016.

[19] S.-N. Yang, C.-H. Ke, Y.-B. Lin and C.-H. Gan, "Mobility management through access network discovery and selection function for load balancing and power saving in software-defined networking environment", EURASIP J. Wireless Commun. Netw., vol. 2016, no. 1, pp. 204, Dec. 2016.

[20] W. Wang, M. Dong, K. Ota, J. Wu, J. Li and G. Li, "CDLB: A cross-domain load balancing mechanism for software defined networks in cloud data centre", Int. J. Comput. Sci. Eng., vol. 18, no. 1, pp. 44-53, 2019.

[21] B. Kang and H. Choo, "An SDN-enhanced load-balancing technique in the cloud system", J. Supercomput., vol. 74, no. 11, pp. 5706-5729, Nov. 2018.

[22] Z. Chen, S. Manzoor, Y. Gao and X. Hei, "Achieving load balancing in high-density software defined WiFi networks", Proc. Int. Conf. Frontiers Inf. Technol. (FIT), pp. 206-211, Dec. 2017.

[23] M. Priyadarsini, J. C. Mukherjee, P. Bera, S. Kumar, A. H. M. Jakaria and M. A. Rahman, "An adaptive load balancing scheme for software-defined network controllers", Comput. Netw., vol. 164, Dec. 2019.

[24] F. Al-Tam and N. Correia, "On load balancing via switch migration in software-defined networking", IEEE Access, vol. 7, pp. 95998-96010, 2019.

[25] Y. Wang, Y. Zhang and J. Chen, "SDNPS: A load-balanced topic-based Publish/Subscribe system in software-defined networking", Appl. Sci., vol. 6, no. 4, pp. 91, Mar. 2016.

[26] M. Farhoudi, P. Habibi and M. Sabaei, "Server load balancing in software-defined networks", Proc. 9th Int. Symp. Telecommun. (IST), pp. 435-441, Dec. 2018.

[27] S. Zhang, J. Lan, P. Sun and Y. Jiang, "Online load balancing for distributed control plane in software-defined data center network", IEEE Access, vol. 6, pp. 18184-18191, 2018.

[28] U. Mahlab, P. E. Omiyi, H. Hundert, Y. Wolbrum, O. Elimelech, I. Aharon, et al., "Entropy-based load-balancing for software-defined elastic optical networks", Proc. 19th Int. Conf. Transparent Opt. Netw. (ICTON), pp. 1-4, Jul. 2017.

[29] X. He, Z. Ren, C. Shi and J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking in the Internet of vehicles", China Commun., vol. 13, pp. 140-149, Nov. 2016.

[30] K. S. Sahoo, M. Tiwary, B. Sahoo, R. Dash and K. Naik, "DSSDN: Demand-supply based load balancing in software-defined wide-area networks", Int. J. Netw. Manage., vol. 28, no. 4, pp. e2022, Jul. 2018.

[31] G. Li, T. Gao, Z. Zhang and Y. Chen, "Fuzzy logic load-balancing strategy based on software-defined networking", Proc. Int. Wireless Internet Conf., pp. 471-482, 2017.

[32] F. Cimorelli, F. D. Priscoli, A. Pietrabissa, L. R. Celsi, V. Suraci and L. Zuccaro, "A distributed load balancing algorithm for the control plane in software defined networking", Proc. 24th Medit. Conf. Control Autom. (MED), pp. 1033-1040, Jun. 2016.

[33] C. Wang, B. Hu, S. Chen, D. Li and B. Liu, "A switch migration-based decision-making scheme for balancing load in SDN", IEEE Access, vol. 5, pp. 4537-4544, 2017.

[34] K. S. Sahoo, D. Puthal, M. Tiwary, M. Usman, B. Sahoo, Z. Wen, et al., "ESMLB: Efficient switch migration-based load balancing for multi-controller SDN in IoT", IEEE Internet Things J.

[35] P. Wang, S.-C. Lin, and M. Luo, “A framework for QoS-aware traffic classification using semi-supervised machine learning in SDNs,”in Proc. IEEE SCC, San Francisco, CA, SA, Jun./Jul. 2016,pp. 760–765.

[36] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, “An efficient elephant flow detection with cost-sensitive in SDN,” in Proc. IEEE INISCom, Tokyo,Japan, Mar. 2015, pp. 24–28.

[37] P. Amaral et al., “Machine learning in software defined networks: Data collection and traffic classification,” in Proc. IEEE ICNP, Singapore,Nov. 2016, pp. 1–5.

[38] Y. Li and J. Li, “MultiClassifier: A combination of DPI and ML for application-layer classification in SDN,” in Proc. IEEE ICSAI,Shanghai, China, Nov. 2014, pp. 682–686.

[39] D. Rossi and S. Valenti, “Fine-grained traffic classification with Netflow data,” in Proc. ACM IWCMC, Caen, France, 2010,pp. 479–483.

[40] Z. A. Qazi et al., "Application-awareness in SDN," in Proc. ACM SIGCOMM, Hong Kong, 2013, pp. 487–488.

[41] A. Nakao and P. Du, "Toward in-network deep machine learning for identifying mobile applications and enabling application specific network slicing," IEICE Trans. Commun., vol. E101.B, no. 7,pp. 1536–1543, 2018.

[42] M. Uddin and T. Nadeem, "TrafficVision: A case for pushing software defined networks to wireless edges," in Proc. IEEE MASS, Brasilia,Brazil, Oct. 2016, pp. 37–46.

[43] Rupani, Kunal, et al. "Dynamic load balancing in software-defined networks using machine learning." Proceeding of International Conference on Computational Science and Applications: ICCSA 2019. Singapore: Springer Singapore, 2020.