Egyptian Informatics Journal 22 (2021) 417–432

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Anomaly Detection and Bottleneck Identification of The Distributed Application in Cloud Data Center using Software–Defined Networking

Ahmed M. El-Shamy [a,*], Nawal A. El-Fishawy [b], Gamal Attiya [b], Mokhtar A. A. Mohamed [b]

[a] Business Technology Department, Canadian International College CIC, Cairo, Egypt
[b] Computer Science and Engineering, Faculty of Electronic Engineering, Menoufia University, Menouf, Egypt

ARTICLE INFO

Article history:
Received 30 September 2020
Revised 18 December 2020
Accepted 5 January 2021
Available online 20 January 2021

Keywords:
Cloud data center network
Software-defined networking
Anomaly detection
Bottleneck identification
Machine learning
Distributed application
Support vector machine

ABSTRACT

Cloud computing applications have grown rapidly in the last decade, where many organizations are migrating their applications to cloud data center as they expected high performance, reliability, and the best quality of service. Data centers deploy a variety of technologies, such as software-defined networks (SDN), to effectively manage their resources. The SDN approach is a highly flexible network architecture that automates network configuration using a centralized controller to overcome traditional network limitations. This paper proposes an SDN-based monitoring algorithm to detect the performance anomaly and identify the bottleneck of the distributed application in the cloud data center using the support vector machine algorithm. It collects the data from the network devices and calculates the performance metrics for the distributed application components that are used to train the SVM algorithm and build a baseline model of the normal behavior of the distributed application. The SVM model detects performance anomaly behavior and identifies the root cause of bottlenecks using one-class support vector machine (OCSVM) and multi-class support vector machine (MCSVM) algorithms. The proposed method does not require any knowledge about the running applications or depends on static threshold values for performance measurements. Simulation results show that the proposed method can detect and locate the failure occurrences efficiently with high precision and low overhead compared to statistical methods, Naive Bayes Classifier and the decision tree machine learning method.

1. Introduction

Cloud computing (CC) depends on data center (DC) networks to connect servers and storage systems using different network devices such as switches, routers, firewalls, and load balancers. Cloud computing hosts various types of distributed applications such as search engines, social networking services, financial services, high-performance computing, and big data analytics applications [1]. Distributed applications run on multiple servers to offer high scalability, reliability, and performance to support an increased workload capacity. These applications generate a large volume of traffic that must be managed efficiently by the servers that process the requests and the network that carries the traffic. As a result, the application performance can be affected and degraded if the Cloud data center (CDC) resources are poorly managed and may lead to overloaded servers or congested networks.

Over the last decade, individuals, companies, and organizations have increasingly relied on cloud computing to run various applications that continue to grow in size and complexity. Application developers and cloud operators are looking for a solution to monitor applications, detect performance anomalies, and identify bottlenecks [2]. It is difficult to monitor and troubleshoot distributed systems because the potential problems can be due to different reasons, including hardware failures, software misconfiguration, application bugs, or network problems [3].

Most web application architecture consists of multiple distributed organized in tiers that interact with each other and exchanging enormous volumes of data over the CDC network infrastructure. Thus, the application performance is highly dependent on the network performance connecting the application components. The deployment of the distributed applications on the CDC is a challenging task, so the design and implementation

* Corresponding author.
E-mail addresses: ahmed_elshamy@cic-cairo.com (A. M. El-Shamy), nelfishawy@hotmail.com (N. A. El-Fishawy), gamal.attiya@yahoo.com (G. Attiya), mokhtar.mohamed@el-eng.menofia.edu.eg (M. A. A. Mohamed).

Peer review under responsibility of Faculty of Computers and Artificial Intelligence, Cairo University.

https://doi.org/10.1016/j.eij.2021.01.001

phase must consider both the application workloads and the DC infrastructure configuration.

To host a distributed application on the CDC, the cloud computing tenant rents a group of virtual machines (VMs) from the cloud provider with the required numbers and hardware specifications according to his design objectives as high performance and availability [4]. There are several options to deploy the distributed application on the CDC as follows.1) A simple configuration would have only one VM for each tier, as shown in Fig. 1-a. 2) A cluster of two or more VMs for the front-end web server tier and using the load balancer to distribute the incoming load among them to support a large number of requests, while a single VM for the other tier, as shown in Fig. 1-b. 3) A cluster of two or more VMs for both the front-end web server and the application server tier and using load balance to distribute the load among them, and only one VM for the database tier, as shown in Fig. 1-c. 4) A cluster of two or more VMs for each tier, including the database tier and implement replication between the database VMs to increase the availability and reliability, as shown in Fig. 1-d.

As a result of the above implementation scenarios, the distributed applications design is complicated and can suffer from performance bottlenecks in any tier of its components.

Cloud providers should optimize application performance and avoid bottlenecks between its components. However, they don't know how these VMs configured, what runs in them, how they are logically connected, and finally, what is the application requirement. Simultaneously, the cloud provider has more options to allocate these VMs in the actual physical servers as set them on the same server or placing them in different servers to reduce the impact of physical server crashes. The VM can be migrated from the current physical server to another server for energy saving or maintenance reasons [5].

According to recent studies [6], the intra-data center traffic has continuously increased with incredible growth compared to the inter-data center traffic and user-facing traffic. The intra-data center traffic is generated between servers and distributed application components, introduces a large load to network devices. The cloud operator must have monitoring and management tools to avoid any performance degradation, enhance the Quality of Service (QoS) of applications, and increase resource utilization. The main functions of the network monitoring tools include detecting detrimental performance behaviors, identify bottleneck's potential root causes, and finally take corrective actions to meet the strict performance requirements according to service level agreements (SLA) items with the tenants [7]. The DC network must meet the various application requirements, such as low latency, high data rate, and bandwidth guarantee for the efficient application's operation [8].

The biggest challenge for cloud service providers is to detect performance anomalies and commit to the required QoS for various applications running on a DC. The applications have different characteristics and varying loads. Distributed application components are deployed on multiple physical and virtual machines. They can be resized or moved through the DC network, making it hard to locate the source of performance problems [5]. One of the most famous challenges for web application developers and businesses is the adequate sizing of their infrastructure to deliver the required QoS by their tenants. So, monitoring the DC infrastructure and analyzing performance metrics is very important to discover abnormal behavior and ensure application requirements can be met by providing different levels of priority, bandwidth, and latency, as required by the application [9].

Cloud data centers based on software-defined networking (SDN) are increasingly accepted and offer many benefits, including network programmability, efficient use of resources, reduced operational costs, better management of the network, and encouragement of innovation [10]. SDN-based Network monitoring solutions are fast, flexible, and scalable without compromising network performance by carefully selecting the traffic flows to monitor by sending instructions to the switches using the OpenFlow protocol [11,12].

Effective performance monitoring of the distributed application provides many invaluable benefits such as 1) keep track of the running application performance to ensure that the business-critical applications are performing as expected and delivered to the end-users with the required quality. 2) useful in troubleshooting and diagnostics of application problems and network configuration error by understanding the previous and current status, investigate the error, and taking appropriate actions to avoid future faults or recover quickly from a failure.3) Improve the capacity planning process to determine future resource requirements for the DC infrastructure.4) Enhance network security and increase the reliability of distributed systems. 5) Billing purpose and financial accounting services [13].

The main contribution of this paper includes an algorithm to discover the performance anomaly behavior and identify bottlenecks in the distributed application using SDN. The proposed method monitors the application and network infrastructure performance in real-time without prior knowledge about the running application or the need for application instrumentation. The SDN controller collects data from various network switches and calculates the following performance metrics: (1) for the application, we calculate response time, throughput, and session number per application tier, (2) for the network, we calculate packet loss, delay, and available bandwidth per network link.

(1) Application and network performance metrics used to train the SVM algorithm and build a baseline model of the normal behavior of the distributed application on DC. The proposed algorithm detects performance anomaly behavior and identifies the root cause of bottlenecks in two steps.
(2) Anomaly detection using the one-class support vector machine (OCSVM) algorithm to classify the front-end server performance by tracking its response time and decides whether the application behavior is normal or abnormal.

Identify the root cause of the bottlenecks using the multi-class support vector machine (MCSVM) algorithm that decides the problem is due to network problems (packet loss, delay, or bandwidth) or an end-host problems. The algorithm enables selective monitoring only on the switch ports related to this application traffic to minimize the collected data and accelerate the identification process. Finally, the algorithm notifies the administrator of the source of the bottleneck to take corrective action.

The rest of this paper is organized as follows. Section 2 provides an overview of SDN technology and application performance detection methods. Section 3 presents some previous work about anomaly detection methods of the distributed application. Section 4 describes the design of the proposed algorithm. Section 5 discusses and analyzes the simulation results. Finally, Section 6 concludes the paper.

2. Background

This section provides an overview about SDN technology, and application performance detection methods.

2.1. Software-Define Networking

The SDN is a highly flexible network management technology to enable a more automated provisioning and policy-based

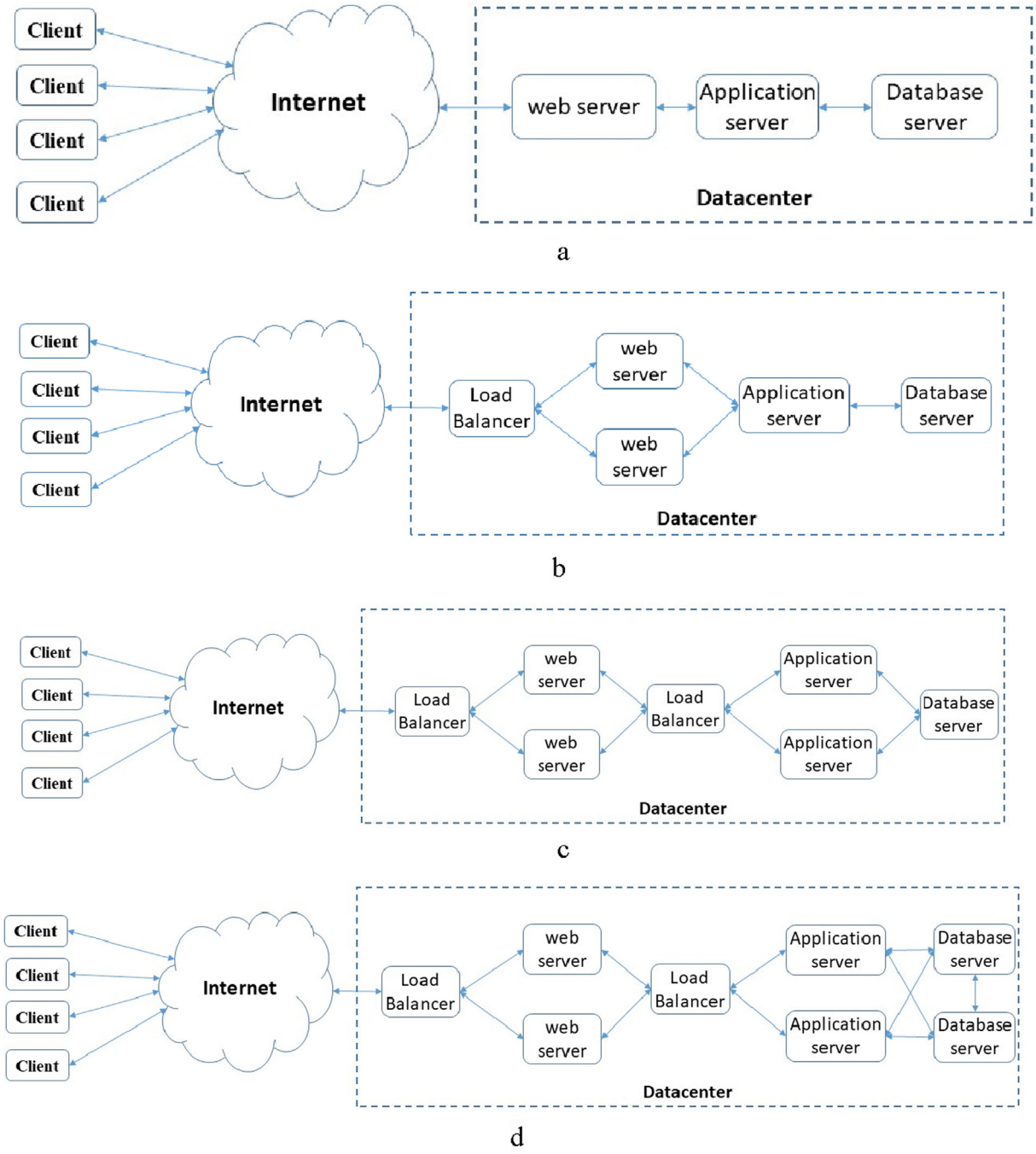

Fig. 1. Distributed application deployment scenarios.

management of network resources [13]. It separates the network control plane of network devices from the underlying data plane that forwards network traffic. The SDN controller is the core element in the SDN technology that centrally manages the network devices to forward the traffic flow using the packet header information, as MAC address, IP address, port number, protocol type, and much other information. While in traditional network architecture, individual network devices make traffic decisions using the destination IP address for routing packets between IP networks and the destination MAC address for switching packets in LAN networks [14].

The SDN framework consists of three layers [15] as shown in Fig. 2. (1) The infrastructure layer, also called the data plane, contains the network devices as switches and routers. It forwards the network packet according to the rules received from the controller. (2) The control layer is the SDN network brain that manages all the network devices centrally by installing rules into the forwarding table using the OpenFlow protocol. (3) The application layer defines the network behavior and communicates with the controller using the northbound API interface as Representational State Transfer [16] to send instructions and retrieve information from the controller.

To manage the DC network efficiently, the SDN controller has a global view of all network components. It establishes a connection with the network devices via the southbound interface, as OpenFlow protocol, to monitor and collect information about network topology, and links utilization [17].

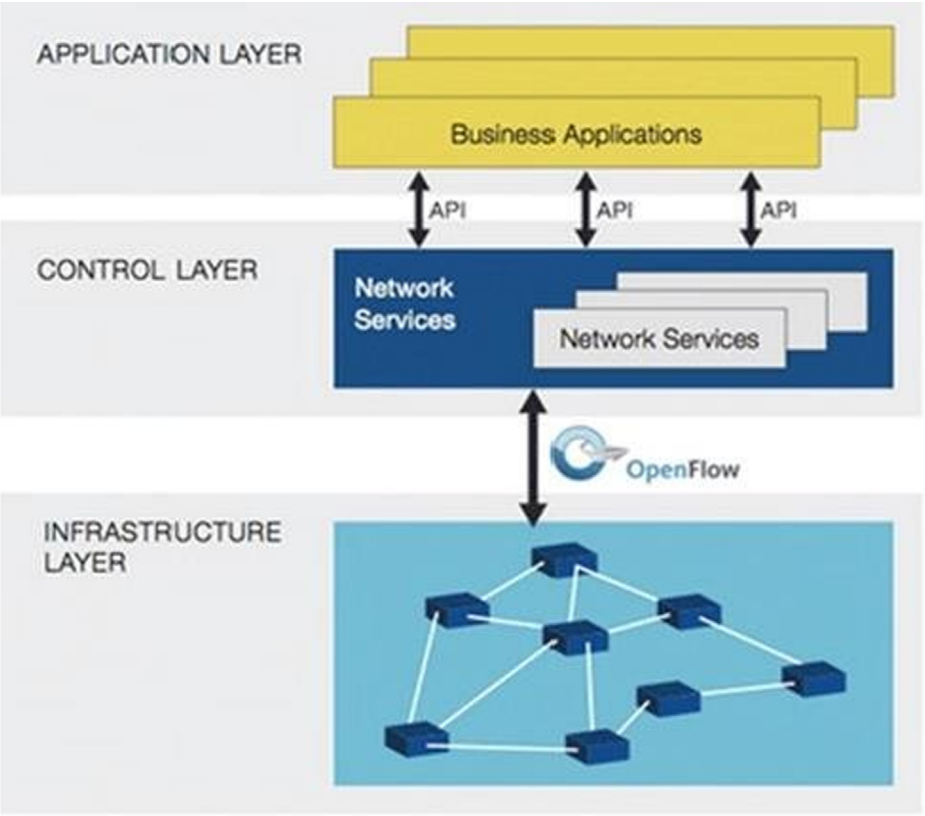

Fig. 2. SDN Framework [10].

2.2. Distributed Application Performance Anomaly Detection Methods

Effective distributed application monitoring solutions require data collection from DC's infrastructure and performance metrics analysis to get useful performance indicators. Existing monitoring approaches fall into two main categories: network-based solutions and host-based solutions, as discussed in the next section.

2.2.1. Network-based Monitoring

Monitoring of applications using network traffic relies on capturing traffic information and analyzing it to assess its performance. These methods are classified into two techniques [18]. (1) Passive monitoring methods observe the transmitted traffic across network nodes. These methods do not affect network performance as they do not inject additional network traffic and are commonly used in the operational data center network. However, they need to install multiple monitoring points that are difficult to implement, and it may take a long time to discover silent applications. (2) Active approaches inject packets into the network or modify the packets to monitor the performance. This additional load affects the accuracy of the measurements and may cause network overload. This method has a lower false-positive rate; however, it cannot be used in the production DC because it may degrade its performance [19].

2.2.2. Host-based Monitoring

Host-based monitoring methods can be classified into (1) application instrumentation methods inject a software code at the application level to monitor and collect application performance metrics [20]. (2) System monitoring methods install a software agent at the operating system level to collect performance metrics about system resources such as processor, memory, hard drive, and network traffic [21].

While host-based monitoring methods are more accurate than network-based methods, the instrumentation process may affect end-host performance and make it less practical. Moreover, this method is hard to implement in the DC network, as it will modify the source code of the application [22].

2.3. Application Performance Anomaly Detection Techniques

There are many methods to detect performance anomaly, including statistical analysis, machine learning, information-theory [23], and k-nearest neighbor-based techniques [24]. The first two techniques are commonly used in most previous research and are relevant to this paper topic, so the next two sections will describe both.

2.3.1. Statistical Analysis

Statistical methods observe system behaviors over time and analyze the relationship between the data to build a model to understand the underlying system dynamics [25,26]. Statistical methods can be classified into two categories: (1) parametric techniques are based on some characteristics of the collected data known as a priori. Examples of these methods are; Tukey's method and Pearson's correlation [2]. Non-parametric techniques are based on a few assumptions about the collected data, such as the histogram used to estimate the data distribution [27].

2.3.2. Machine Learning

There are three types of machine learning methods: (1) Supervised learning methods use labeled data during the training phase, which contain both normal and anomalous data. Examples of these

methods are support vector machine, decision tree, regression, and Naive Bayes. While these methods achieve high accuracy, they cannot detect unknown anomalies that are not present in the training data. It is hard to get training data that contains all anomalous types [28]. (2) Semi-supervised learning methods combine a small amount of labeled data with a large amount of labeled data during the training. (3) Unsupervised learning methods develop a model to describe the hidden structure of unlabeled data. It does not require any labeled data for training. Principal component analysis, association rule mining, and clustering methods are examples of this method. These methods are suitable for the practical environment because of the lack of label data, but they are less efficient than supervised methods [29].

3. Related Work

This section presents an overview of network monitoring using the SDN that has been studied over the last decade as explained in the next part.

Liu et al. [30] proposed NetAlytics, a network monitoring framework for cloud data centers using SDN. Their system collects and aggregates network traffic in a processing engine to analyze the application's response time using the Apache Storm data analytics engine. But there are some comments on this search, including that they did not define the location and required number of the monitoring system and processing engine. The proposed algorithm did not present an algorithm to calculate the average or a reference value to diagnose the application's performance. Finally, they did not suggest any actions or give recommendations to improve the low-performance application- tier.

Chowdhury et al. [31] focused on the trade-off between monitoring accuracy, timeliness, and network overhead. The proposed model supports various monitoring objectives like performance, fault-tolerance, and security. But their model did not correlate between the network and the performance metrics of the running application. They did not suggest a method to calculate the normal value of the performance metrics to detect anomalies.

Fu et al. [32] proposed a monitoring approach to measure packet loss rates. Meanwhile, Yuan et al. [33] monitored the link utilization by combining fuzzy logic with OpenFlow messages to observe network traffic and obtain flow statistics. Megyesi et al. [34] measured the available bandwidth of the link between any two switches. These solutions focus mainly on individual flows or packets without analyzing the relationship between them.

Peng et al. [35] proposed a flow detection method based on the SDN to detect and classify distributed denial of service (DDoS) attacks using the double P-value of the transductive confidence machines for the K-nearest neighbor algorithm. Granby et al. [36] proposed a centralized SDN-based platform to detect the DC anomalies and mitigate the limitations of existing distributed monitoring techniques by reactively identifying threats in real-time.

Suarez-Varela and Barlet-Ros [37] proposed a scalable flow monitoring and classification solution for OpenFlow switches using a sampling-based method. For the monitoring process, the SDN controller installs a set of rules in the network switches to enable traffic flow sampling. The DC switches match the incoming packets to check whether they are part of the flow monitoring entries. If it matches, the packet counters are updated. The flow classification method uses a combination of deep packet inspection and machine learning techniques, focusing on web and encrypted traffic identification. Elsaadawy et al. [38] proposed a network monitoring approach using custom port detection techniques and compare the different switch port mirror methods like a port mirror, selective mirror, tunnel mirror, and truncated mirror using quantitative comparison.

Van Adrichemet et al. [39] proposed a QoS monitoring method for network flow, called OpenNetMon, by collecting data at the source and destination switches using an adaptive rate sampling. They calculate the delay, throughput, and packet loss to decide whether QoS requirements are met or not. Siniarski et al. [40] proposed a lightweight monitoring solution based on SDN, named FlowVisa, to identify network flows that belong to the critical applications by interacting directly with them. The proposed method can analyze traffic flows and discover the switches that serve these flows.

Tahari et al. [41] proposed a synchronization mechanism for aggregating traffic flow statistics from distributed SDN controllers that manage the DC networks. The proposed solution consists of two layers; the first layer collects flow statistics from the network switches and sends them to a coordinator, in the second layer, to aggregate the flow information.

Liu et al. [42] presented OpenMeasure as a flow measurement and inference framework with continuous online learning to track the most informative network flows. They proposed two online learning algorithms for designing adaptive flow measurement rules: an algorithm based on weighted linear prediction and another algorithm that adopts the strategy used in multi-armed bandit problems [43].

Lazaris and Prasanna [44] proposed DeepFlow, a traffic measurement framework based on the SDN. They install a set of rules in the free memory space (TCAM) in the switches to measure the critical traffic flow. The Long Short-Term Memory- Recurrent Neural Network "LSTM-RNN" machine-learning algorithm predicts the size of the flows that cannot be monitored with installed rules using historical data from the previous measurement.

Yang and Yeung [45] proposed the lonely flow first (LFF) algorithm to monitor network flows that pass through only a single switch to minimize the bandwidth consumed by monitoring traffic. They divide the DC switches into, with or without the lonely flows. Also, they used a weight function to decide the polling order and cost of flow polling and defined the distance and the message overhead as two communication costs to decide whether to use poll-single or poll-all switches. The proposed method compared the cost of each switch group to decide the polling method, poll-single messages method are used if the poll-all is a higher cost and vice versa until all flows are covered.

Garget al. [46] proposed a hybrid anomaly detection system based on deep learning to detect suspicious flows in social multimedia applications. It consists of two modules: (1) an anomaly detection module based on the Restricted Boltzmann Machine (RBM) and the support vector machine (SVM) to detect the abnormal activities, (2) an end-to-end data transmission algorithm to meet the QoS requirements for multimedia applications as high bandwidth and low latency.

Tang and Haque [47] proposed a resilient monitoring framework named ReMon that can efficiently recover from link failure by merging the network packet measurement with the aggregated network statistics to improve the measurement accuracy and minimize the measurement cost. Their solution consists of three algorithms. (1) The Weight Assisted Selecting (WAS) algorithm selects a group of switches to be monitored to minimize the monitoring cost and polls the flow statistics from those switches using the sFlow protocol [48]. (2) Anchor Assisted Recovery (AAR) and Weight Assisted Recovery (WAR) algorithms and integrated them in the ReMon framework to provide measurement resiliency in the case of link failure.

Rezende et al. [49] proposed SDNMon as an extension module of the SDN controller to monitor the network devices in the data plan and improve the control plan information about the network topology. SDNMon can monitor the bandwidth and latency per port and flow. SDNMon exploits threads to collect selected port

and flow statistics using the sFlow protocol and polling mechanism, based on network topology.

Shen [50] proposed a monitoring method based on SDN to observe a selected group of the OpenFlow switches to reduce resource consumption. The proposed method consists of two phases. 1) The monitoring phase selects the monitored switches according to the following steps: scans all network switches, counts their flows, sorts them according to their number of flows, and selects switches with the highest number of flows. At the end of this phase, the covered flows are removed from the unselected network switches. 2) The flow re-routing phase tries to combine switches with the lowest re-routing cost to re-route the flows covered by the selected network switches.

Wang and Su [51] proposed a flexible SDN-based monitoring framework called FlexMonitor to detect the DDoS attacks by using selective monitoring strategies that select both the switch and end-host pair selection. The FlexMonitor consists of four modules: 1) the monitoring module interprets the upper management application requests and chooses the appropriate monitoring strategy based on application needs. 2) The monitoring strategy deployment module carries out network monitoring by deploying a specific monitoring strategy. 3) The collection module collects the monitoring data accurately from the network switches and hosts periodically. 4) The analysis module aggregates and analysis of the collected data.

Xing et al. [52] proposed a FlowMon based on a sample-and-fetch- mechanism to detect large flows. It consists of two stages. 1) The sampling stage uses packet sampling to detect elephant flows. 2) The counting stage uses the flow table counting method to determine the largest flow among the suspicious flows. Afeket al. [53] proposed a Sample and Pick algorithm to detect large flow in the SDN network. It consists of two parts; the first part is the sampling methods that define the rate for creating counting rules in the switches and the packets sampling rate using different packet sampling methods. The second part focuses on large flow detection methods.

Madanapalli et al. [54] designed a solution to protect the DSN controller. It monitors the traffic flows on the network links then a software inspection engine receives and inspects the packets to protect the SDN controller from overload.

Cohen and Moroshko [55] proposed a sampling-on-demand monitoring framework that allows the SDN controller to set the sampling rate of each flow rate at each switch as the network operator sets this rate according to the monitoring goal. I proposed a sampling-on-demand monitoring framework in which the SDN controller selects the sampling rate of each flow at each switch as the network operator sets this rate according to the monitoring goal. Their proposed framework consists of three components. The first component, the Sampling Management Module, is deployed in the SDN controller that samples the flows based on the specified rate. The second component, the Sampling Module, is implemented in the network switches. The third component is the Collecting Server collects and processes the sampled packets. Their framework defines a new OpenFlow message called "OFPT RATE MOD" sent by the Sampling Management Module to the network switches to set the sampling rate for each flow.

Santos et al. [56] proposed an approach to collect network statistics using OpenFlow messages to measure QoS metrics (e.g., utilization of port bandwidth and loss) and make them available for upper applications such as traffic engineering. These messages are sent periodically within a predefined polling interval (i.e., every 5 s). In OpenTM [57] single query is issued during each polling interval (5 s) for every different source destination pair. They used the source–destination pair to identify flows and calculate the volume of traffic. In [58], the SDN controller periodically extracts the port statistics from network switches every 500 ms to obtain accurate results. The consumed bandwidth between any two neighbor switches can be computed using the transmitted bytes by each network switch. In [59], the controller polls network switches periodically (i.e., every second) to collects queues statistics.

4. Proposed System Architecture

The proposed method evaluates the distributed application performance using the support vector machine technique (SVM) to detect the anomalies and identify the root causes of bottlenecks based on SDN. The SDN controller collects statistics from the SDN switches, and calculates the performance metric for both the distributed application components and network devices. We calculate response time, throughput, and session number per application tier while calculating packet loss, delay, and available bandwidth per network link. We train the SVM algorithm using the performance metrics data and build a baseline model for normal application behavior under varying network and end-host load.

The proposed method monitors the front-end server response time and classifies its performance as normal or abnormal using the one-class support vector machine (OCSVM) algorithm. We select the response time, as it represents the overall performance of the application and the network. If performance is classified as normal, it means that all application components and network devices function efficiently. Otherwise, the performance has deviated from the baseline model. This means that a network device or application component is operating improperly. In this case, we initiate the multi-class support vector machine (MCSVM) algorithm to identify the performance bottleneck and discover the responsible application tier or the network device that degrades the overall application performance. A detailed explanation of all the modules will be covered in the following parts.

4.1. Monitoring Module

The SDN controller can discover the network topology and measure the network parameters while the monitoring node (MN) monitors the application response time and discovers its components. The SDN controller configures the TOR switch to forward a copy of all packets to the MN to collect information about each flow of its rack servers as shown in Fig. 3. The monitoring module operates as a separate virtual machine to minimize the load of the SDN controller and can be started or stopped in any rack if required. The following parts discuss in detail how to calculate the application and network metrics.

4.1.1. Network Topology Discovery

The SDN controller communicates with network devices using the OpenFlow protocol to manage, configure the network devices, and has an up-to-date network topology. The following steps describe how the SDN controller can discover the entire data center infrastructure, including the network devices, how they are connected, and the end-hosts.

4.1.1.1. Discover Switches. The network administrator configures the SDN switches with the IP address and port number of the SDN controller to establish a secure connection via Transport Layer Security (TLS). At startup, the controller can discover the DC network by sending "FEATURE_REQUEST_MESSAGE" to each device, and then the switch responds with "FEATURE_REPLY_MESSAGE" containing its identifier (chassis ID), active ports, and MAC addresses of the connected hosts. Fig. 4 displays a simple SDN network topology with three OpenFlow switches connecting four hosts using the mininet simulation [67]. The controller can dis-

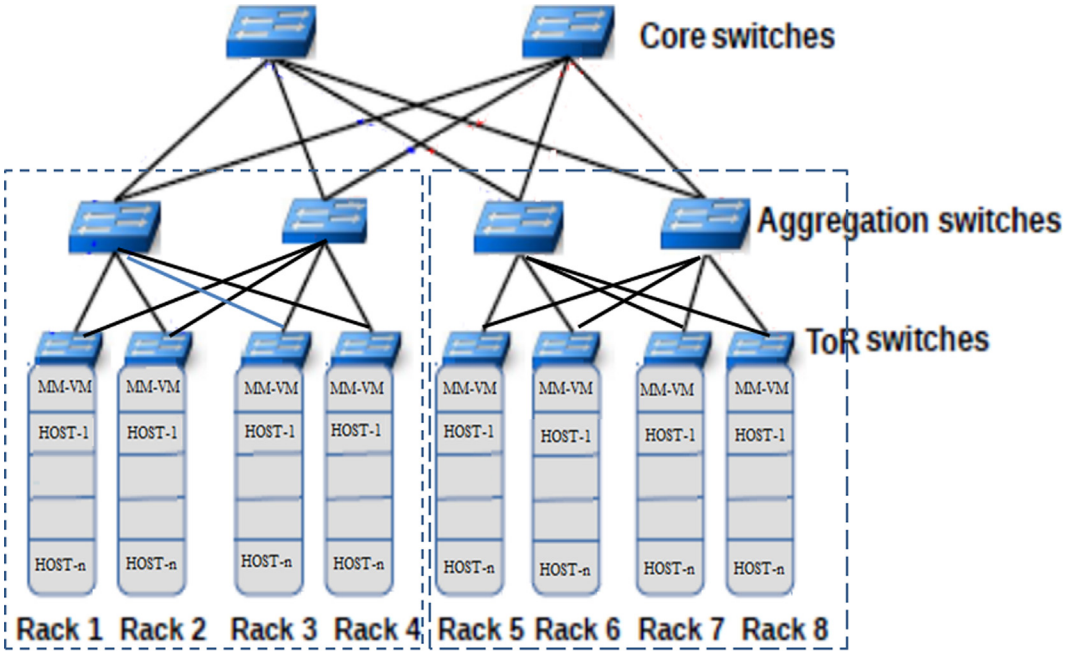

Fig. 3. Monitoring data center networks.

cover the switches S1, S2, and S3 and their active ports, but it cannot detect the inter-switch connectivity links [60].

4.1.1.2. Discover Inter-Switches Links. The SDN controller can discover inter-switch links using OpenFlow Discovery Protocol (OFDP) by creating a separate Link Layer Discovery Protocol (LLDP) packet [61] for each active port on each switch learned in step 1. The LLDP packet contains the switch ID, port number, and time to live (TTL). The controller sends the LLDP packets within the "OFPT_PACKET_OUT" message to each switch with instructions to forward it out of a specific port. When a switch receives the LLDP packet from an adjacent switch, it has a default rule to forward it to the controller within the OFPT_PACKET_IN message; this enables the controller to discover the inter-switch links [60].

When the S1 switch receives "OFPT_PACKET_OUT" packets from the controller, it sends them through all active ports. Then the S2 switch receives this packet and returns it to the controller. This way, the controller can discover the link between the S1 and S2 switches. This process is repeated in the S2 and S3 switches to discover the connection between them.

4.1.1.3. Hosts Discovery. The controller can discover the distributed application components within the network and the switches that connect them when a host starts sending packets to the network.

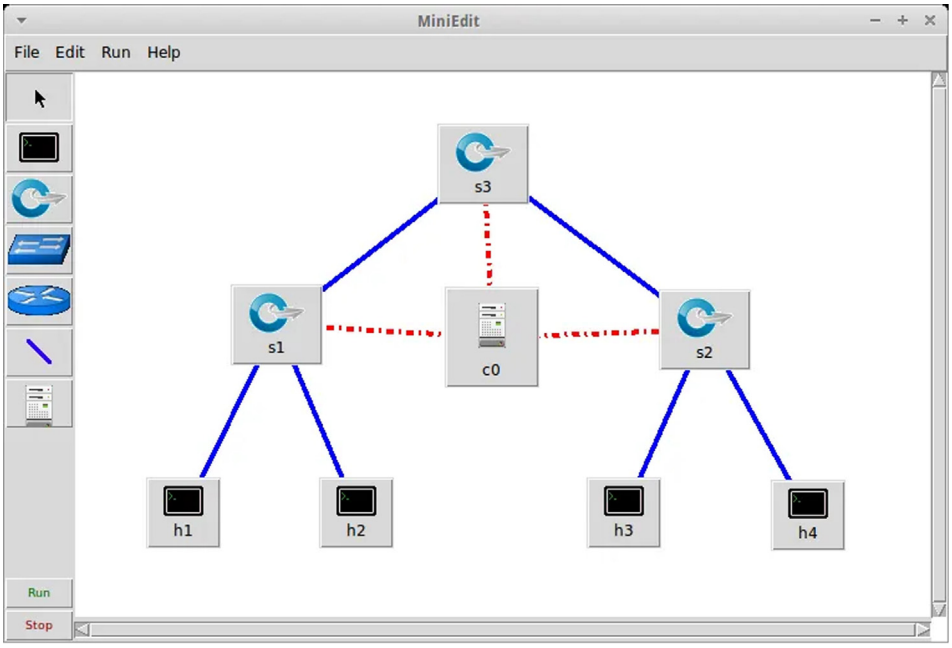

Fig. 4. Simple SDN network using the mininet simulation.

The switch encapsulates the first packet in the "OFPT_PACKET_IN" message and forwards it to the controller to request a new rule. This way, the controller can identify the connected hosts to any switches. The "OFPT_PACKET_IN" contains host-related information as the IP address, MAC address, and the switch port number that connects the host [60].

4.1.2. Network Parameters Measurement

This part explains the steps to measure network performance metrics, including latency, bandwidth, and packet loss.

4.1.2.1. Calculate Latency. The link latency ($L_{s1\text{-}s2}$) for the link between the S1 and S2 switches, as shown in Fig. 4, can be calculated using the following equation:

$$L_{s1-s2} = t_2 - t_1 - (RTT_{c-s1}/2) - (RTT_{c-s2}/2) \quad (1)$$

where t_1 is the transmission time of the "OFPT_PACKET_OUT" packet from the controller to the S1 switch as explained in step (Discover Inter-Switches Links), t_2 is the receiving time of the "OFPT_PACKET_IN" packet by the controller from the S2 switch. While RTT is the round trip time of the channel between the controller and any switch, it can be calculated using the timestamp in the "OFPT_ECHO_REQUEST" and "OFPT_ECHO_REPLY" messages.

$$RTT_{c-s1} = t_{r1} - t_{s1} \quad (2)$$

$$RTT_{c-s2} = t_{r1} - t_{s1} \quad (3)$$

where $RTT_{c\text{-}s1}$and$RTT_{c\text{-}s2}$ are the round-trip time between the controller and the S1 and S2 switches respectively, t_{s1} and t_{s2} are the timestamp of the "OFPT_ECHO_ REQUEST" message sent by the controller to the S1 and S2 switches respectively, t_{r1} and t_{r2} are the timestamp of the "OFPT_ECHO_REPLY" packet sent by the S1 and S2 switches to the controller respectively.

4.1.2.2. Calculate Available Link Bandwidth. The available bandwidth BWs1-s2 of the link Ls1-s2 connecting the S1 and S2 switches, as shown in Fig. 4, can be calculated using the "STATISTICS _REQUEST" message sent by the controller to the switch requesting its ports' statistics and the "STATISTICS_REPLY" reply message sent by the switch to the controller as illustrated in the following equations [60].

$$BW_{s1-s2} = Port\_Speed - -Consumed\_Bandwidth \quad (4)$$

$$Consumed\_Bandwidth = (TR_{t2} - TR_{t1})/(t2 - t1) \quad (5)$$

where TR_{t1} and TR_{t2} are the data transmitted through the switch port at time t_1 and t_2 respectively.

4.1.2.3. Calculate Packet Loss. The packet loss (PL) ratio over a link between any two switches can be calculated by polling the flow statistics of the source and destination switches, by subtracting the increase of the transmitted packet counter (TX) in the source switch with an increase of the received packet counter (RX) of the destination switch at two different times t1, t2 as in Eq. (6) [62].

$$PL = 1 - \frac{RX_{s2}(t2) - RX_{s2}(t1)}{TX_{s1}(t2) - TX_{s1}(t1)} \quad (6)$$

4.1.3. Application Parameters Measurement

This part explains the steps to calculate the application performance metrics from a network perspective, including application throughput, application response time, and application workload session calculated per each tier of the distributed application.

4.1.3.1. Application Throughput. The application throughput refers to the flow size within a time interval. It can be calculated using the "STATISTICS_REQUEST" message sent by the controller to a switch requesting information about a specific traffic flow and the "STATISTICS_REPLY" reply message by a switch to the controller with the required information statistics. For each flow, the throughput can be calculated using the number of packets (S) during the sampling interval (T) as in eq. (7).

$$\text{Throughput} = S/T \text{ byte/sec} \quad (7)$$

4.1.3.2. Application Response Time. The application response time breakdown per-tier can be calculated by inspecting the TCP packet headers and checking the TCP SYN/ACK/FIN flags that indicate the start and the end of the TCP session between application tiers. Here in this paper, we focus on the TCP traffic. The TCP session has three phases: (1) connection establishment to set up the connection using the three-way handshake messages, (2) data transfer, (3) connection tear down to terminate the connection using the four-way handshake messages. Then the response time per tier can be calculated approximately by the time interval between the SYN and FIN messages as shown in Fig. 5.

4.1.3.3. Application Workload. The application workload refers to the number of concurrent sessions to the server that hosts the application tier. It can be calculated by applying a rule to the TOR switch to select only all the flows where this server is the destination and then count the number of flows per time interval.

4.2. Analyzing Module

This module proposed a dynamic baseline model to evaluate the application performance metrics instead of using a static and fixed threshold value to classify the application performance. We build the baseline model using the SVM model and train it using the performance metrics data. The performance metrics data is collected using a varying workload to simulate the real data center as load changes from time to time.

Performance metrics (M) can be represented as a collection of (n) metrics of the network and application as in Eq. (8).

$$M = \{m_i |\ n \geqslant i \geqslant 1\} \quad (8)$$

For each metric m_i at time j, the algorithm uses the sliding window S_{ij} with the size t containing a set of collected values v_{ij} as in eq. (9)

$$S_{ij} = v_{ij},\ v_{i(j+1)},\ \ldots..,\ v_{i(j+t)} \quad (9)$$

The collection of n metrics S_{ij} describes the distributed application behavior AB_j at time j during the time interval t as in Eq. (10)

$$AB_j = \{ S_{ij}1 \leqslant i \leqslant n \} \quad (10)$$

Feature extraction techniques are applied to the collected dataset ABj to derive some relevant properties using statistical analysis. This technique minimizes the calculation overhead, increases the algorithm accuracy, minimizes the storage space, and reduces the dataset dimension from t to d_k while preserving the time series characteristics.

The set of features S_f that are extracted from S_{ij} can be represented as in Eq. (11)

$$S_f = \{ f_x |\ 1 \leqslant i \leqslant z \} \text{ where } f_x : R^t \rightarrow R^{d_x} \quad (11)$$

Data normalization is the last step in data processing before feeding data to the SVM algorithm. As there are different ranges of feature vector values and the SVM assumes that the data it

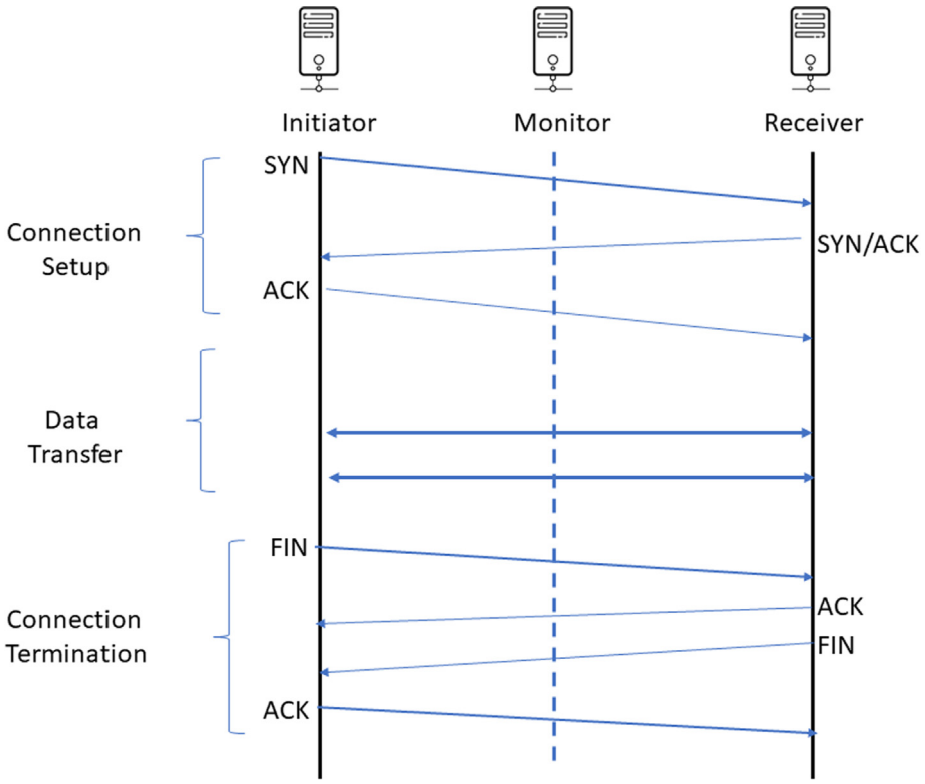

Fig. 5. The response time measurement using TCP flags.

works with are in a standard range (0 to 1) or (−1 to 1). Therefore, using sigmoid function that transform the input data to the output range of 0–1 using the following Eq. (12)

$$x_i' = \frac{1}{1+e^a} \quad where\ a = \frac{x_i - x_{mean}}{x_{std}} \tag{12}$$

The SVM classifier training phase finds the optimal hyperplane that maximizes the margin between two classes using a set of labeled vectors L

$$L = \{(x_i;\ y_i) \mid 1 \leqslant i \leqslant n\},\ \text{where}\ x_i \in R^n\ and\ y_i = \{-1,\ 1\} \tag{13}$$

Finding the optimal hyperplane requires minimizing $||w||^2$, where w is a vector normal to the hyperplane that can be expressed as the optimization problem as in Eq. (14)

$$\sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j x_i.x_j\ subject\ to\ \alpha_i \geqslant 0\ and\ \sum_{i=1}^{l}\alpha_i y_i = 0 \tag{14}$$

where α_i and α_j are Lagrange multipliers.

For the non-linear model, a kernel trick is used as it is more efficient and less expensive way to transform the data into higher dimensions by replacing $(x_i.x_j)$ with the kernel function as in Eq. (15)

$$\sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j \varphi(x_i).\varphi(x_j)\ subject\ to\ \alpha_i \geqslant 0\ and\ \sum_{i=1}^{l}\alpha_i y_i = 0\ Where\ \varphi : R^n \to R^m \tag{15}$$

The radial basis function (RBF) kernel function is widely used to build the hyperplane in the m-dimensional space to separate the dataset categories. RBF is defined as in Eq. (16)

$$K(x, x') = \exp - \gamma \| x - x' \|^2 \tag{16}$$

The SVM decision function classifies the unlabeled vectors x_j according to their position with respect to the hyperplane using the following Eq. (17)

$$f(x_i) = sgn\left(\sum_{i=1}^{l}\alpha_i y_i x_i + b\right) \tag{17}$$

4.3. Performance Anomaly Detection and Bottleneck Identification Module

The performance anomaly detection and bottleneck identification module consists of two steps using the SVM algorithm [63]:

1- The One-class support vector machine (OCSVM) classifies the response time performance of the front-end server as normal or abnormal.
2- The multi-class support vector machine (MCSVM) identifies the type of anomalies and determines the root cause of the bottlenecks.

The following pseudo code for algorithm 1 describes the proposed Anomaly Detection and Bottleneck Identification algorithm:

Algorithm 1. *Distributed Application Performance Anomaly Detection and Bottleneck Identification*

Input: Flow information collected from the switches using the SDN controller
Output: Detect the performance anomaly, identify the root cause of the bottlenecks of distributed application
Steps:
1- SDN controller discover the network topology
2- SDN establish a session with each network devices using OpenFlow protocol
3- SDN controller collect OpenFlow statistics from the DC switches
4- SDN build the DC network topology
5- for each TOR switch in network
6- SDN Enable port mirroring to forward traffic to the monitoring node (MN)
7- MN starts collecting flow information
8- Discover the distributed application components
9- Calculate network performance metrics (latency, available BW and packet loss)
10- Calculate application performance metrics (throughput, response time and workload)
11- If network status is stable (steady state)
12- Then
13- Disable port mirror
14- Else
15- Port mirror still enabled
16- End if
17- End for
18- for each distributed application in the CDC
19- calculate the application performance metrics
20- build baseline model Using SVM techniques
21- Train the SVM model using vary workload
22- End for
23- detection mode
24- for each front-end server in distributed application
25- monitor the response time
26- Classify the front-end server performance using OCSVM model
27- if application response is normal
28- Then
29- The application performance comply with SLA
30- Keep monitor the front-end server
31- else
32- the application response is abnormal
33- Set the robustness factor C
34- if Count the number of consecutive detection $\geq$ C (robustness factor)
35- then
36- Enable selective monitoring only in switch ports that are related to this application
37- Identify bottleneck reason using MCSVM model: network or end host problem
38- Alert administrator
39- End if
40- End for

Algorithm 1 of the distributed application performance anomaly detection and bottleneck identification works as follows.

At the startup of the algorithm, the SDN controller discovers the network, establishes a session with each network device, collects the data from the DC switches using OpenFlow protocol, and finally builds an up-to-date DC network topology in lines (1–4). The SDN controller enables the port mirror in each TOR switch to forward the traffic to the monitoring node (MN) and starts collecting information about the application flow to discover the components of the distributed application in lines (5–8). Based on collected data about network and application, the algorithm calculates the network performance metrics, including latency, available BW, and packet loss for each network device. Also, the algorithm calculates the application performance metrics, including throughput, response time, and workload for each component of the distributed application in lines (9–10). Lines (11–13) disable the port mirror setting after collecting all the required information and discover all application components. This way we minimize the overhead in TOR switches.

If there are changes in the network and a new application is implemented, the port mirror is still active to track the new changes in the DC network, as in the lines (14–17). Using the calculated metrics for the performance of distributed applications, we build a baseline model using SVM techniques for all running distributed applications per tier. This model is trained with data representing different workloads to simulate the real DC network, as in the lines (18–22). The anomaly detection module monitors the response time performance of the front-end server in distributed applications. It classifies its performance as normal or abnormal using the one-class support vector machine (OCSVM) algorithm. If application response is normal and complies with SLA then we continue to monitor only the performance of the front-end server, as in lines (23–30).

If abnormal behavior is detected, the algorithm waits to count the same anomaly for the number C of consecutive detections before considering it as an anomaly. This step is important to reinforce the robustness of the proposed algorithm against false alarms. Variable C is called the robustness factor, increasing the value of C increases the detection time, but minimizes the false alarm ratio. In the experimental section, we define the optimal value of C which maximizes the robustness of the algorithm as in the lines (31–34). If an anomaly is detected, we start a selective monitoring process in switch ports only that relate to this application to further investigate the behavior of each tier and find out the reason for this bottleneck, whether it's a network issue or a host issue. The MCSVM model is used to determine the root cause of bottlenecks. Finally, sending an alert to the network administrator indicates the type of defect, as in tines (35–39). The overall system architecture is shown in Fig. 6.

5. Simulation Evaluation and Results Analysis

This section presents the evaluation of the proposed anomaly detection approach.

5.1. Testbed Setup

To simulate the multi-tier web application, we deploy the RUBiS application in a three-tiered architecture where we install its components on three separate machines; each has 4 GB of RAM and 2 CPU cores. Apache 2.0.54 is installed on the web server, apache tomcat 8.5.51 is installed on the application server, and the database server is MySQL 8. On each machine, we install an OVS switch to act as a TOR switch. The SDN switch connects the distributed application machines, the SDN controller, and one client machine, as shown in Fig. 7. The SDN controller used in the simulation is the OpenDaylight (ODL). It is installed on a separate machine with 2 GB of memory and 2 CPU cores. ODL is an open source written in Java and used to automate networks of any size [64]. Monitoring and analyzing roles are implemented in the SDN controller for simplicity. The port mirroring is enabled on the SDN switch ports to send a copy of the RUBiS application component traffic to the controller.

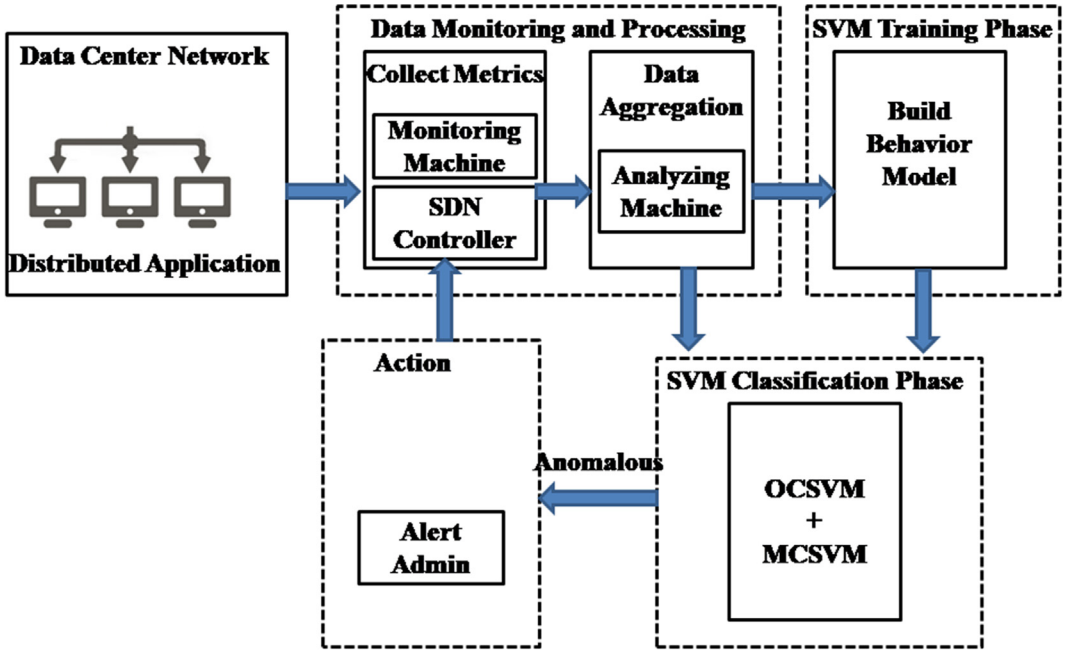

Fig. 6. The overall system architecture.

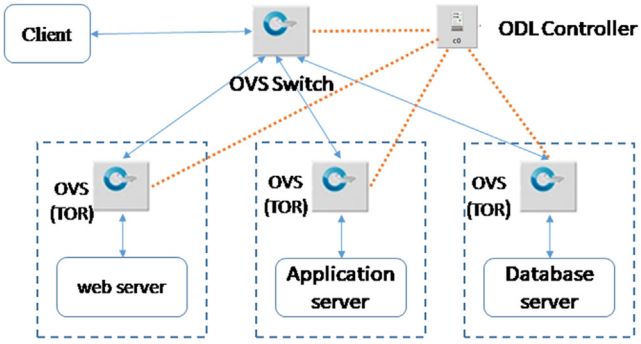

Fig. 7. Testbed setup.

5.2. Testing Scenarios

The training phase has two objectives: First, create a baseline model that describes the normal performance behavior with a varying load on the network and end-host. Second, train the SVM algorithm to detect performance anomalies, where we train the algorithm to discover five types of anomalies: response time, packet loss, delay, bandwidth, and end-host problems. Based on the following test scenarios, the duration of each test is 120 min and repeated 20 times:

1) Run the distributed application under normal conditions with a varying network and end-hosts workload to simulate normal data center environment activities.
2) Increase the packet loss ratio in the network link to cause a network bottleneck at time 30, 60, 90 min in different network links
3) Repeat the test number 2 by increasing the delay and changing the bandwidth of network links at the same specified time intervals
4) Increase the processor utilization ratio using a PHP software script to consume the CPU intensively to degrade the end-host performance. We run this software script randomly on the various components of the distributed application.

5.3. Evaluating Metrics

The main objectives of the proposed algorithm include detect the performance anomalies as soon as possible, identify their location, and finally, send an alert to the system administrator. The anomalies detection must be carried out with a minimum number of false alarms generated by the algorithm. The typical metrics used to evaluate the anomaly detection system are the true-positive rate (TPR) and the false-positive rate (FPR). TPR is the rate at which the system can correctly identify the positive cases, and FPR is the rate at which the system incorrectly predicts the positive cases [65]. Based on these metrics, we deploy Recall, Precision, and F-SCORE to measure the proposed model's accuracy.

The Recall or Confidence Score is the ratio of correctly detected anomalies to all anomalous instances as in eq. (18).

$$\text{Recall R} = \frac{\text{detected anomalies (TP)}}{\text{total number of anomalies (TP + FN)}} \tag{18}$$

Precision: the ratio of the correctly detected anomaly to the sum of the correctly and incorrectly detected anomalies as in Eq. (19).

$$\text{Precision P} = \frac{\text{detected anomalies (TP)}}{\text{total number of alerts (TP) + (FP)}} \tag{19}$$

The F-score is the harmonic mean of Recall and Precision as in Eq. (20).

$$F = 2\frac{P * R}{\text{P} + \text{R}} \tag{20}$$

The classification accuracy is the total number of correct predictions divided by the total number of predictions. As a performance measure, accuracy is inappropriate for imbalanced classification problems. The main reason is that the overwhelming number of the majority class will overwhelm the number of the minority class, meaning that even unskillful models can achieve accuracy scores of 90 percent depending on how severe the class imbalance happens to be. An alternative to using classification accuracy is to use precision and recall metrics.

5.4. Results and Discussion

5.4.1. The Effect of Window Size on Anomaly Detection

Here we test the effect of window size on the performance of the anomaly detection using SVM and compare it against fixed threshold value methods based on two different approaches: standard deviation (SD) and mean absolute deviation (MAD) as in many previous researches [66].

The window size t of the sliding time series, discussed in part 4.2, has a large effect on the F-score. The F-score value is computed using the SVM model and two fixed threshold value methods using the SD and MAD statistical methods, as shown in Fig. 8. The results indicate that when the window size value is small, especially less than 45 s, the value of the F-score is decreased because the classifier model cannot capture the pattern of the time series data. Fig. 8 also indicates that the shortage of fixed threshold methods compared to the machine learning method due to the presence of outliers and abnormal values that cause a significant bias in the threshold value.

5.4.2. The Effect of Robustness Factor

To avoid misclassification by the SVM model due to a few consecutive abnormal values, we define a robustness factor C, which assists the SVM model in the anomaly detection decision-making process. The model classifies the application performance as abnormal, if it observes C consecutive abnormal values. As shown in Fig. 9, the robustness factor value increases as the false alarm rate decreases. Fig. 10 shows the robustness factor against the missed detection rate, the percentage of the anomalous values that are identified as normal as robustness factor. According to the results of these two cases as shown in Figs. 9 and 10, we find the optimal value of the robustness factor is 6 to obtain a lower false alarm rate and to keep the anomaly miss rate low.

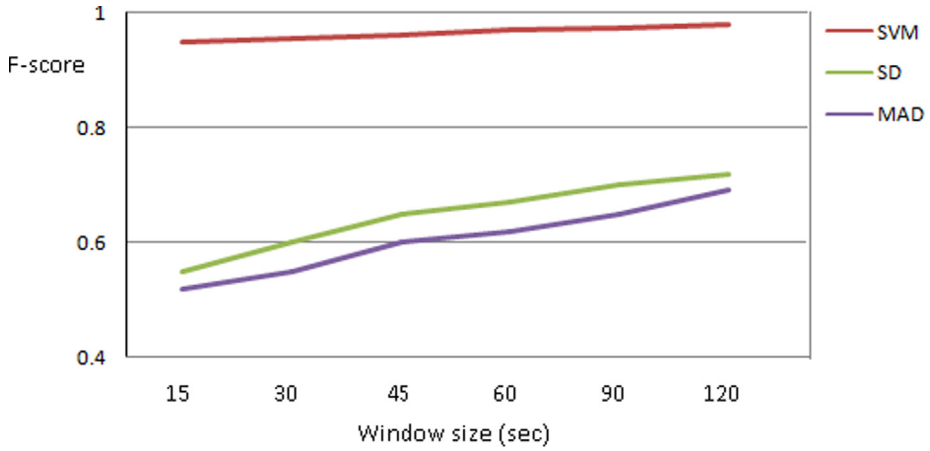

Fig. 8. The effect of window size on the F-score.

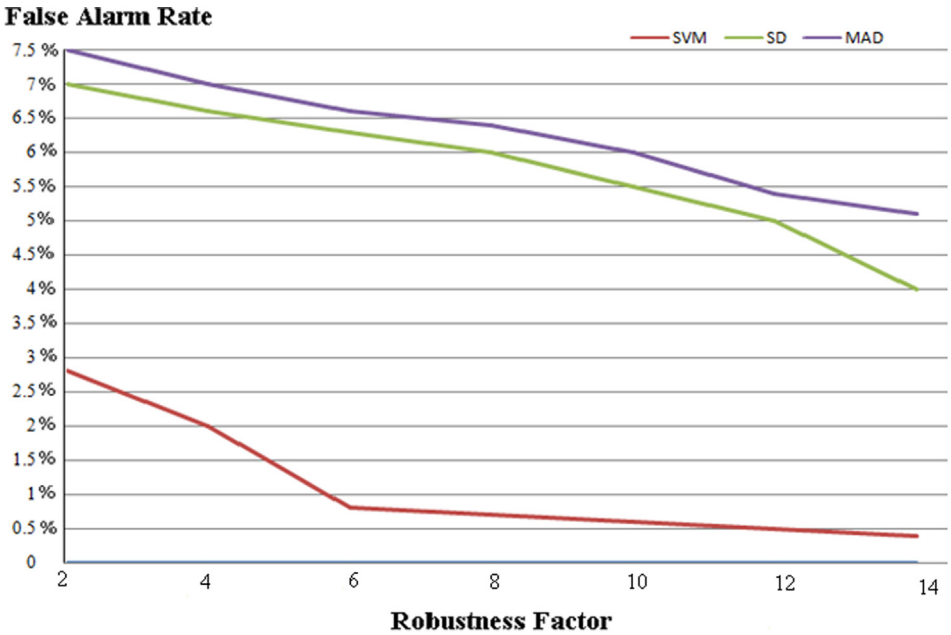

Fig. 9. The effect of robustness factor on the false anomaly alarm.

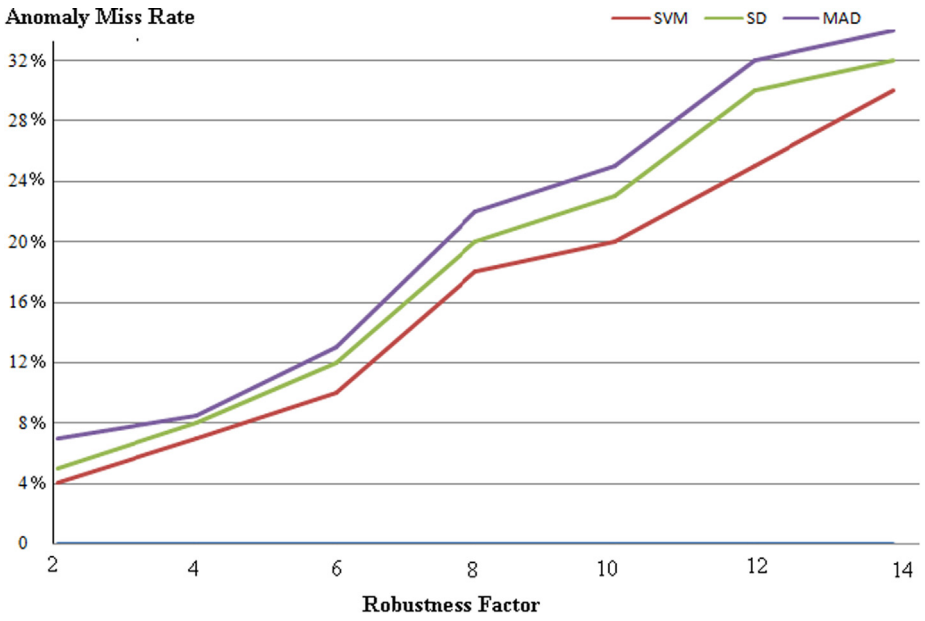

Fig. 10. The effect of robustness factor on the anomaly miss rate.

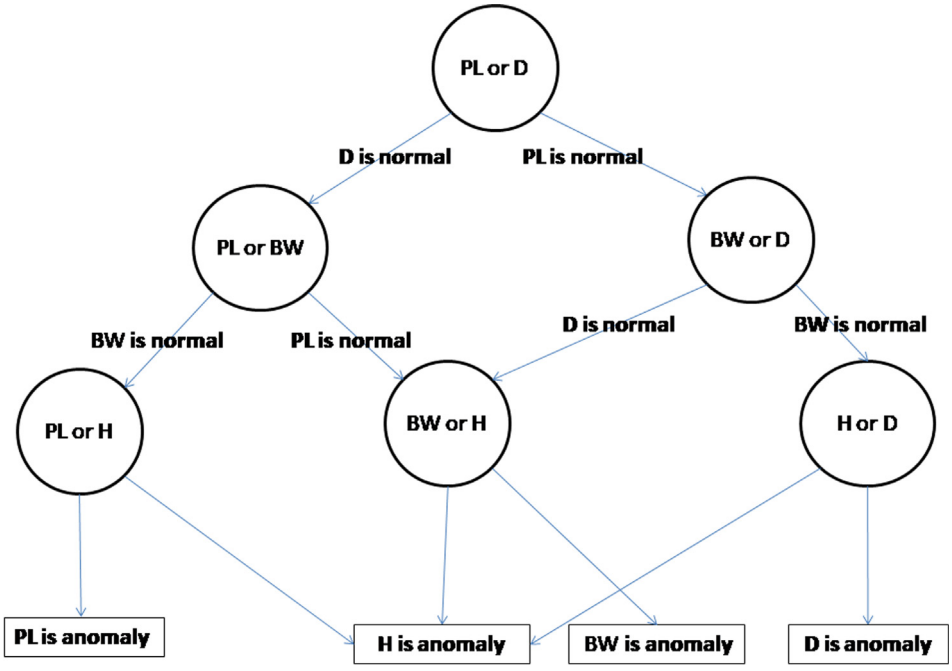

Fig. 11. Decision tree of multi-class SVM based on binary classifier.

5.4.3. Identify the Bottleneck

According to the training phase, we train the system with a five types of anomalies affecting the following performance metrics at different application layers: response time (RT), packet loss (PL), delay (D), and available bandwidth (BW) and CPU of end-host (H). During the run-time, we inject an increase in packet loss at time 25 min, an increase in the delay at time 50 min, a decrease in bandwidth at time 75 min, and high CPU usage at time 100 min. Each anomaly lasts for 10 s.

The proposed algorithm can identify the type of bottleneck using MC-SVM which converts this classification to multi binary classifiers organized in a decision tree as in Fig. 11 and the results of bottleneck identification in Fig. 12, the proposed algorithm can detect bottleneck in less than 7 sec as it must wait for six consecutive anomalies values (robustness factor C = 6) to consider it as anomaly.

5.4.4. Classification Performance Comparison

In this part, we evaluate the classification performance to detect the anomaly of distributed applications using three different machine learning algorithms, including SVM, Naive Bayes (NB),

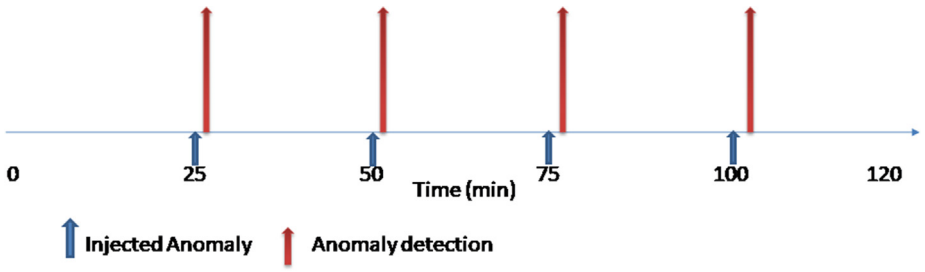

Fig. 12. The anomaly detection process.

Table 1
The performance of the proposed method in comparison with other techniques.

Performance Measure	SVM	NB	DT	LR
Recall	96.31	88.647	92.54	82.11
Precision	95.54	89.77	94.81	83.2
F-score	98.5	93.54	97.3	85.15

the decision tree (DT) and one statistical model, the logistic regression (LR), using the calculated performance metrics.

The classification performance of the proposed approach on the calculated metrics is evaluated using three machine learning classifiers the SVM, Bayes (NB) and the decision tree (DT). As can see in Table 1, we observed that the SVM performs better than the other two techniques

Area Under Curve (AUC) metric

We define performance metric area under curve (AUC) which represents an aggregate measure of performance across all possible classification thresholds, the area represent the classification performance, the more the area, the better is the classifier, When AUC is equal to 1, the classifier achieves perfect accuracy if the threshold is correctly chosen. The following Eqs. (21)–(23) explain how to calculate AUC

$$fpr = \frac{fp}{fp + TN} \tag{21}$$

$$\text{Specificity} = 1 - FPR \tag{22}$$

$$AUC = \frac{\text{Specificity} + \text{Recall}}{2} \tag{23}$$

The results shown in Fig. 13 indicate that the SVM classifier has the highest AUC, and it has the best performance in distinguishing between normal and abnormal behavior compared to other methods.

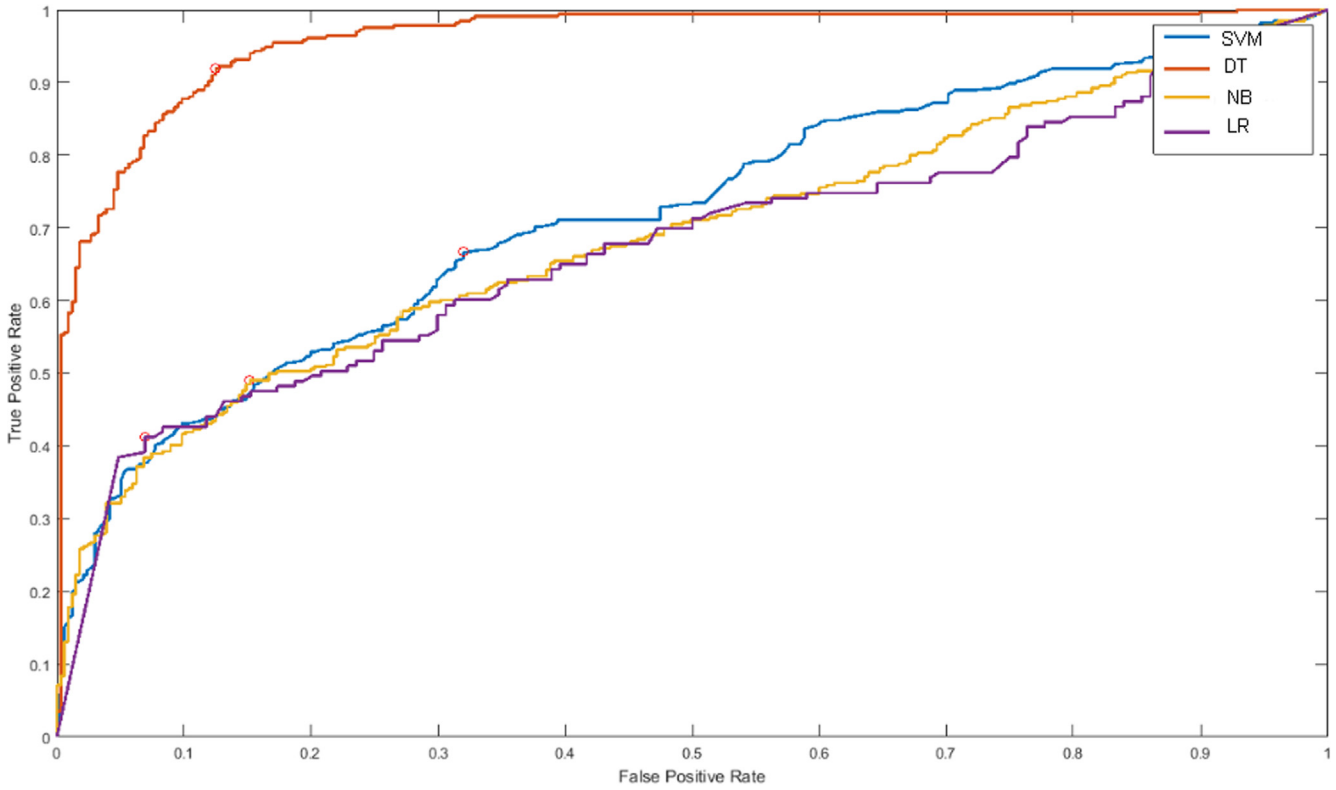

Fig. 13. The area under the curve.

6. Conclusion

In this paper, we proposed a new SDN-based monitoring algorithm to detect the performance anomaly and identify the bottleneck of the distributed application in the cloud data center using the support vector machine algorithm. The SDN Controller collects data from network switches and calculates performance metrics for distributed application components and network devices. These performance metrics used to train the SVM algorithm and build a baseline model of the normal behavior of the distributed application on DC. The SVM model detects performance anomaly behavior and identifies the root cause of bottlenecks in two steps. (1) The One-class support vector machine (OCSVM) classifies the response time performance of the front-end server as normal or abnormal. (2) The multi-class support vector machine (MCSVM) identifies the type of anomalies and determines the root cause of the bottlenecks.

The proposed algorithm tries to minimize monitoring overload on the network devices using selective monitoring techniques. Finally, the algorithm notifies the administrator of the source of the bottleneck to take corrective action. The benefits of the proposed method, including it has a lower false alarm rate and monitors the performance of the datacenter infrastructure in real time without prior knowledge about the running application or the need for application instrumentation.

For future work, we plan to increase the collected metrics of the data center infrastructure and mainly focus on end-host parameters. Extend this solution by testing it to monitor the big data processing application like Map Reduces framework. Additional machine learning models will be included in the comparison.

References

[1] I. Odun-Ayo, M. Ananya, F. Agono, and R. Goddy-Worlu, "Cloud computing architecture: A critical analysis," 18th International Conference on Computational Science and Applications (ICCSA), Melbourne, VIC, pp. 1–7, 2018.

[2] Rath M. Resource provision and QoS support with added security for client side applications in cloud computing. Int J Inform Technol 2019;11(2):357–64.

[3] Li X, Li K, Ding Y, Wei D, Ma X. Application of autonomous monitoring method based on distributed environment deployment in network fault. J Phys Conf Ser 2020;1486:022048.

[4] Deng S, Xiang Z, Taheri J, Mohammad KA, Yin J, Zomaya A, et al. Optimal application deployment in resource constrained distributed edges. IEEE Trans Mob Comput 2020.

[5] Syed HJ, Gani A, Ahmad RW, Khan MK, Ahmed AIA. Cloud monitoring: a review, taxonomy, and open research issues. J Netw Comput Appl 2017;98:11–26.

[6] M. Akter, M. M. S. Maswood, S. S. Sonia and A. G. Alharbi, "A Novel Approach to Reduce Bandwidth Cost and Balance Network and Server Level Load in Intra Data Center Network," 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, pp. 194-198, 2020.

[7] Yadav GHK, Madhavi K. Response time-based resource allocation according to service level agreements in cloud computing. Int J Int Technol Secur Trans 2019;9(4):537–46.

[8] Shirmarz A, Ghaffari A. Performance issues and solutions in SDN-based data center: a survey. J Supercomput 2020:1–49.

[9] S. Varshney, S. Rajinder, and P. K. Gupta. "QoS based resource provisioning in cloud computing environment: a technical survey." International Conference on Advances in Computing and Data Sciences. Springer, Singapore, 2019.

[10] Abbasi, A. Abbasi, S. Shamshirband, A. T. Chronopoulos, V. Persico and A. Pescapè, "Software-Defined Cloud Computing: A Systematic Review on Latest Trends and Developments," in IEEE Access, vol. 7, pp. 93294-93314, 2019.

[11] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, "SuVMF: Software-defined unified virtual monitoring function for SDN-based large-scale networks," in Proceedings of The Ninth International Conference on Future Internet Technologies. ACM, 2014, p. 4.

[12] Y. Yu et al., "Fault Management in Software-Defined Networking: A Survey," in IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 349–392, Firstquarter 2019, doi: 10.1109/COMST.2018.2868922.

[13] Karakus M, Durresi A. Quality of service (QoS) in software defined networking (SDN): a survey. J Netw Comput Appl 2017;80:200–18.

[14] S. Clayman, L. Mamatas and A. Galis, "Efficient management solutions for software-defined infrastructures," NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, pp. 1291–1296, 2016.

[15] E. Haleplidis, S. Denazis, Kostas Pentikousis, J. Hadi Salim, D. Meyer and O. Koufopavlou, "Software-defined networking (SDN): Layers and architectures terminology", RFC7426, 2015.

[16] W. Zhou, L. Li, M. Luo and W. Chou, "REST API design patterns for SDN northbound API," 28th International Conference on Advanced Information Networking and Applications Workshops, Victoria, BC, pp. 358-365, 2014.

[17] Bakhshi Taimur. State of the art and recent research advances in software defined networking. Wireless Commun Mobile Comput 2017.

[18] Binsahaq A, Sheltami TR, Salah K. A survey on autonomic provisioning and management of QoS in SDN networks. IEEE Access 2019;7:73384–435.

[19] Dai B, Xu G, Huang B, Qin P, Xu Y. Enabling network innovation in data center networks with software defined networking: a survey. J Netw Comput Appl 2017;94:33–49.

[20] O. Vysocký, L. Říha, and A. Bartolini. "Application instrumentation for performance analysis and tuning with focus on energy efficiency." Concurrency and Computation: Practice and Experience, 2020.

[21] J. Hwang, G. Liu, S. Zeng, F. Y. Wu and T. Wood, "Topology Discovery and Service Classification for Distributed-Aware Clouds," 2014 IEEE International Conference on Cloud Engineering, Boston, MA, pp. 385-390, 2014.

[22] Tsai P, Tsai C, Hsu C, Yang C. Network monitoring in software-defined networking: a review. IEEE Syst J Dec. 2018;12(4):3958–69.

[23] Neelakanta Perambur S. Information-theoretic aspects of neural networks. CRC Press; 2020.

[24] G. Muruti, F. A. Rahim, and Z. Ibrahim. "A survey on anomalies detection techniques and measurement methods." IEEE Conference on Application, Information and Network Security (AINS). IEEE, 2018.

[25] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning," 2017.

[26] Sha W, Zhu Y, Chen M, Huang T. Statistical learning for anomaly detection in cloud server systems: a multi-order markov chain framework. IEEE Trans Cloud Comput 2018;6(2):401–13.

[27] Zugazagoitia E, Queral C, Fernández-Cosials K, Gómez J, Durán LF, Sánchez-Torrijos J, et al. Uncertainty and sensitivity analysis of a PWR LOCA sequence using parametric and non-parametric methods. Reliab Eng Syst Saf 2020;193.

[28] S. He, J. Zhu, P. He and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, , pp. 207-218, 2016.

[29] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, pp. 1285–1298, 2017.

[30] G. Liu, M. Trotter, Y. Ren, and T. Wood. "NetAlytics: Cloud-Scale Application Performance Monitoring with SDN and NFV". In Proceedings of the 17th International Middleware Conference, Trento, Italy, pp. 1-14, 2016.

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

[32] C. Fu, W. John and C. Meirosu, "EPLE: An Efficient Passive Lightweight Estimator for SDN packet loss measurement," 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, pp. 192-198, 2016.

[33] X. Yuan and F. Hu, "OFMON: An Adaptive Flow Monitoring Framework for SDN," 2017 International Conference on Network and Information Systems for Computers (ICNISC), Shanghai, China, pp. 37-41, 2017.

[34] P. Megyesi, A. Botta, G. Aceto, A. Pescaṕe, and S. Molńar, "Challenges and solution for measuring available bandwidth in software defined networks," in: Computer Communications, Elsevier B.V, vol. 99, pp. 48–61, 2017.

[35] Peng H, Sun Z, Zhao X, Tan S, Sun Z. A detection method for anomaly flow in software defined network. IEEE Access 2018;6:27809–17.

[36] R. Granby, B. Askwith and A. K. Marnerides, "SDN-PANDA: Software-Defined Network Platform for Anomaly Detection Applications," 2015 IEEE 23rd International Conference on Network Protocols (ICNP), San Francisco, CA, pp. 463-466, 2015.

[37] Suarez-Varela J, Barlet-Ros P. Flow monitoring in Software-Defined Networks: Finding the accuracy/performance tradeoffs. Comput Netw 2018;135:289–301.

[38] M. Elsaadawy, B. Kemme and M. Younis, "Enabling Efficient Application Monitoring in Cloud Data Centers using SDN," ICC 2020 - 2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, pp. 1-6, 2020.

[39] N. L. Van Adrichem, C. Doerr and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," IEEE Network Operations and Management Symposium (NOMS), Krakow, pp. 1-8, 2014.

[40] B. Siniarski, J. Murphy, and D. Delaney, "FlowVista: Low-bandwidth SDN monitoring driven by business application interaction," in Proc. 25th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM), pp. 1-6 2017.

[41] Tahaei H, Salleh RB, Razak MFA, Ko K, Anuar NB. Cost effective network ow measurement for software de_ned networks: a distributed controller scenario. IEEE Access 2018;6:5182–98.

[42] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in Proc. IEEE INFOCOM, pp. 47_52, 2016.

[43] Auer P, Cesa-Bianchi N, Fischer P. Finite-time analysis of the multiarmed bandit problem. Mach Learn 2002;47(2–3):235–56.

[44] A. Lazaris, and V. K. Prasanna. "DeepFlow: a deep learning framework for software-defined measurement." Proceedings of the 2nd Workshop on Cloud-Assisted Networking. 2017.

[45] Ze Yang and K. L. Yeung, "An efficient flow monitoring scheme for SDN networks," 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, pp. 1-4, 2017.
[46] Garg S, Kaur K, Kumar N, Rodrigues JJPC. Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in SDN: a social multimedia perspective. IEEE Trans Multimedia 2019;21(3):566–78.
[47] F. Tang and I. Haque, "ReMon: A Resilient Flow Monitoring Framework," 2019 Network Traffic Measurement and Analysis Conference (TMA), Paris, France, pp. 137-144, 2019.
[48] "sFlow-RT," https://sflow-rt.com/index.php.
[49] Rezende PHA, Coelho PRSL, Faina LF, Camargos LJ, Pasquini R. Analysis of monitoring and multipath support on top of OpenFlow specification. Int J Network Manage 2018;28:e2017.
[50] S.-H. Shen, "An efficient Network Monitor for SDN Networks," ACM SIGMETRICS Performance Evaluation Review, vol. 46, pp. 95–96, 2019.
[51] Wang B, Su J. FlexMonitor: a flexible monitoring framework in SDN. Symmetry 2018;10(12):713.
[52] Xing C, Ding K, Hu C, Chen M. Sample and fetch-based large flow detection mechanism in software defined networks. IEEE Commun Lett 2016;20:1764–7.
[53] Afek Y, Bremler-Barr A, Landau Feibish S, Schi L. Detecting heavy flows in the SDN match and action model. Comput Netw 2018;136:1–12.
[54] S. C. Madanapalli, M. Lyu, H. Kumar, H. H. Gharakheili and V. Sivaraman, "Real-time detection, isolation and monitoring of elephant flows using commodity SDN system," NOMS 2018 - IEEE/IFIP Network Operations and Management Symposium, Taipei, pp. 1–5, 2018.
[55] Cohen R, Moroshko E. Sampling-on-demand in SDN. IEEE/ACM Trans Network 2018;26:2612–22.
[56] R. B. Santos, T. R. Ribeiro, and C. D. A. César, "A network monitor and controller using only open_flow," in Proc. 8th Latin Amer. Netw. Oper. Manage. Symp. (LANOMS), pp. 9–16, 2015.
[57] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for open_flow networks," in Passive and Active Measurement, Berlin, Germany: Springer, vol. 6032, p. 201–210, 2010
[58] Singh M, Varyani N, Singh J, Haribabu K. Estimation of End-to- End Available Bandwidth and Link Capacity in SDN. Cham, Switzerland: Springer; 2017.
[59] Rowshanrad S, Namvarasl S, Keshtgari M. A queue monitoring system in openflow software defined networks. J Telecommun Inf Technol 2017;1:39–43.
[60] S. Zeng, P. Zheng and Y. Zhang, "Design of test case for openflow protocol conformance test based on OFTest," international symposium on computer, consumer and control (is3c), Xi'an, pp. 465-470, 2016.
[61] P. Congdon, Link Layer Discovery Protocol, RFC 2922, 2002.
[62] W. Aljoby, X. Wang, T. Fu and R. Ma, "On SDN-enabled online and dynamic bandwidth allocation for stream analytics," IEEE 26th International Conference on Network Protocols (ICNP), Cambridge, pp. 209-219, 2018.
[63] Cervantes J, Garcia-Lamont F, Rodríguez-Mazahua L, Lopez A. A comprehensive survey on support vector machine classification: applications, challenges and trends. Neurocomputing 2020.
[64] OpenDaylight https://www.opendaylight.org/.
[65] Y. Tan, X. Gu, and H. Wang, "Adaptive System Anomaly Prediction for Large-Scale Hosting Infrastructures," in Proceedings of the 29th Symposium on Principles of Distributed Computing, pp. 173–182, 2010.
[66] J P Simmons, L D Nelson and U Simonsohn (2011). False positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. Psychological Science, 22(11), 1359–1366.
[67] http://mininet.org/.