Received February 9, 2021, accepted February 18, 2021, date of publication February 23, 2021, date of current version March 5, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3061601

R-EDoS: Robust Economic Denial of Sustainability Detection in an SDN-Based Cloud Through Stochastic Recurrent Neural Network

PHUC TRINH DINH[1], (Member, IEEE), AND MINHO PARK[1,2], (Member, IEEE)

[1]Department of Information Communication, Materials, and Chemistry Convergence Technology, Soongsil University, Seoul 156-743, South Korea

[2]School of Electronic Engineering, Soongsil University, Seoul 156-743, South Korea

Corresponding author: Minho Park (mhp@ssu.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2020R1F1A1076795).

ABSTRACT Cloud computing is now known as the most cost-effective platform for delivering big data and artificial intelligence services over the Internet to enterprises and cloud consumers. However, despite many recent security developments, many cloud consumers continue to express great concern about using these platforms because they still have significant vulnerabilities. Typically, Economic Denial of Sustainability (EDoS) attacks exploit the pay-as-you-go billing mechanisms used by cloud service providers, so that a cloud customer is forced to pay an extra fee for the additional resources triggered by the attack activities. In our previous work, we already proposed an system to mitigate such EDoS attacks. Overall, this previous work presented an effective system for detecting abnormal events; however, the false-alarm rates still remain relatively high and detection rates are low, because abnormal events could be caused by the cloud customer. Furthermore, our previous work still consumes a large number of computing resources. Therefore, in this paper, we propose an enhanced scheme to detect and mitigate EDoS attacks efficiently and reliably. Our proposed scheme is composed of online and offline phases, implementing a gated recurrent unit, which not only can capture complex temporal dependence relations in the data, but also can reduce the vanishing gradient problems in time series. First, to reflect the normal patterns, our proposed scheme learns accurate representations of multivariate time series. Next, these representations are used to reconstruct input data. Finally, the reconstruction probabilities not only can be used to find anomalies, but also can provide interpretations. The proposed scheme also introduces a self-adjusting threshold to reduce error rates, whereas existing solutions normally use a hard threshold to analyze the anomalies, which causes increasing error rates. Our comprehensive analysis of the results shows outstanding performance compared to other solutions and our previous work.

INDEX TERMS Economic denial of sustainability, software-defined networking, deep learning, cloud computing, network function virtualization, service function chaining.

I. INTRODUCTION

In Cloud Computing, two technologies known as software defined networking (SDN) [1] and network functions virtualization (NFV) [2] are quickly becoming core technologies. SDN offers a new method which very significantly reduces the complexity of network devices and provides efficient network management. NFV is designed to develop, deliver, and manage network services much less expensively through virtualization. Moreover, a new technology known as service function chaining (SFC), which is the use of ordered lists of service functions composing virtual chains, chains together network services such as load balancing, antivirus, and deep packet inspection, and then steers the network traffic through such services (Figure 1).

A. PROBLEM STATEMENTS

Cloud computing provides some unique features such as dynamic resource assignment and usage-based payment that are considered to be in high demand compared to

The associate editor coordinating the review of this manuscript and approving it for publication was Xianzhi Wang.

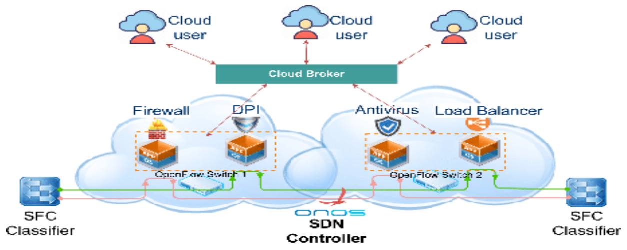

FIGURE 1. SFC sitting on NFV integrating with SDN on cloud.

conventional computing services. However, there remain grave concerns about cloud computing security [5]. For instances, exploiting a pay-as-you-go pricing engine on the cloud can force users to pay more for their resource usage. This is commonly known as an EDoS attack, and is one of the most difficult cloud security challenges.

The definition of EDos attack is as follows. A special feature on the cloud called auto-scaling monitors various metrics such as network bandwidth, memory, and CPU usage. This is triggered when the parameters cross predefined thresholds. Once activated, a cloud service provider automatically allocates more resources to meet the increased demands and charges the cloud user according to the service-level agreement. An EDoS attacker exploits the auto-scaling engine to force a target cloud user to pay more for extra resources. EDoS problems become extremely difficult to tackle because:

- In the use of SFC, virtual network functions (VNFs) are normally public to internet users. Therefore, EDoS attackers exploit this to launch more VNFs or other reasons [7].
- Existing solutions addressing EDoS attacks are mainly hard-threshold-based solutions with high false-alarm rates [6]. Also, they work well only for a certain distribution of attack traffic, i.e. a Poisson distribution. Moreover, they only focus on some specific types of attack, leaving significant remaining vulnerabilities [39].
- There is still a trade-off between resource consumption and detection performance for an EDoS defender. Therefore, it is a difficult challenge to find a solution that satisfies both resource usage and detection performance requirements.

The harmful effects of EDoS attacks are thus a serious security limitation of essentially all cloud-based network systems. Moreover, not many solutions can tackle EDoS attack effectively [6]. A more intelligent, robust approach is needed to ensure that the EDoS attacks can be mitigated effectively.

B. OUR PROPOSAL

We propose a robust, intelligent, and efficient scheme to tackle EDoS attacks, employing a technique named multivariate time series anomaly detection. In the proposed scheme, an anomaly score, used as a basis of attack detection, is compared with a dynamic threshold set automatically according to a trained machine learning model processing traffic statistics collected from a network system. Thus, our scheme can detect EDoS attacks efficiently and is well-adapted to diverse network systems. Furthermore, as a recommendation inspired by N. Agrawal *et al.*, an SDN-based cloud security service provides various options on the cloud (i.e., software-based traffic analysis [47]). Therefore, we propose a more complete EDoS defense system to detect and mitigate the EDoS attacks, which we name robust EDoS defender (R-EDOS).

C. CONTRIBUTION

In summary, our main contributions are listed as follows:

- First, existing solutions addressing EDoS attacks are mainly hard-threshold-based solutions with high false-alarm rates [6]. Also, their solutions work well only for a certain distribution of attack traffic, i.e. a Poisson distribution. Moreover, they only focus on some specific types of attack, leaving significant remaining vulnerabilities [39]. Therefore, we propose a new approach that can tackle the above issues, known as a multivariate time-series, to deal with stealthy EDoS attacks, effectively.
- Secondly, we propose an efficient scheme called R-EDoS which applies the proposed approach to detect anomalous network data generated by EDoS attacks.
- Next, we investigate the characteristics of EDoS attacks, and present common types of EDoS attacks in an SDN-based cloud environment.
- We conducted an experiment using simulated different EDoS attacks on a popular cloud computing platform called OpenStack and an ONOS SDN controller [49]. This integration could be replaced by other SDN controllers, such as OpenDaylight or Floodlight.
- Finally, we collected and analyzed the experimental data, and made a detailed comparison with other algorithms and other existing proposals, including our previous work, in different EDoS attack scenarios. An extensive comparison of R-EDoS against other approaches proves the detection capability of our proposed scheme.

The remainder of this research study is presented as follows. The next section presents related work and Section III discusses background information. Section IV introduces our research rationale, system design analysis, and internal modules. Section V describes our testbed. In Section VI, a variety of evaluation metrics are used to evaluate our proposed scheme. Finally, Section VII draws some conclusions and presents future improvements.

II. RELATED WORK

Recently, EDoS has constantly attracted the attention of researchers for preventing the EDoS attacks from manipulating the auto-scaling engine of the cloud providers. Table 1 shows different approaches to detect and mitigate EDoS attacks.

Even though many proposals have been proposed to tackle EDoS attacks (as presented in Table 1); however, no existing proposals have the right approach and propose an effective scheme to tackle such stealthy EDoS attacks effectively and dynamically.

TABLE 1. Comparison of different EDoS mitigation approaches.

No.	Security Mechanism	Description	Limitations
1	EDoS-Shield [8]	EDoS-Shield is composed of two main components (virtual firewall and authentication nodes).	EDoS-Shield is vulnerable to spoofing attacks. It also produces high false-positive rates due to blocking many legitimate IP addresses.
2	sPoW [9]	sPoW focuses only on mitigating EDoS attacks by filtering illegitimate traffic.	First, sPoW is acutely vulnerable to puzzle accumulation attacks. Second, it poses high false-positive rates because illegitimate users can be denied access to the service request.
3	EDoS Armor [10]	EDoS Armor is composed of the admission control phase and congestion control phase where the former phase restricts end-users in quantities and the latter phase uses a decision tree algorithm and categorizes the client as good and bad users based on their browsing history.	The adaptability of the model should be taken into consideration because the site could be too complex for potential new users to lose interest.
4	EDoS Eye [11]	Game theory is applied to construct a game scenario among two parties (attacker and defender). Also, the Nash equilibrium solution blocks the EDoS attacker.	• Game Theory fails to operate reliably in case of having big payment matrices. • The use of honeypot is unnecessary. If it has to extract new signatures, then it will not be useful for detection. Moreover, automated signature extraction is quite a hard task. • The attack traffic has to follow Poisson distribution so that Game Theory can work well.
5	EDoS mitigation mechanism (EMM) [12]	EMM consists of two detection phases collecting and classifying phases.	• Lack of evaluation metrics. • Attack scenarios are not practical and too simple.
6	EDoS-ADS [13]	This scheme can identify the legitimate clients where they may come from network address translation network. It operates in four different modes which are flash overcrowd, attack, normal, and suspicious.	• Its simulation is not close to real-world scenario [12]. • The attack mitigation rises great cost due to the blacklist rule table.
7	Proactive EDoS [14]	Proactive EDoS uses hard thresholds to detect EDoS attacks. It is deployed on an AWS cloud platform.	• It raises high false-error rates due to the hard thresholds. • The attack traffic has to follow Exponential distribution. • The service rate has to follow the Poisson distribution.
8	Entropy-based EDoS [15]	To detect EDoS attacks, an entropy-based model is proposed.	The proposed work suggests good detection accuracy, however, it experimented in a very simple testbed, which raises doubt about its performance in a real-world network.
9	Fuzzy Entropy EDoS [24]	A feedback mechanism based on fuzzy logic including Fuzzy entropy integrated with Lion Neural Learner is proposed.	Fuzzy Entropy EDoS produces high errors because of the predefined rule. This results in generating high errors.
10	Execution trace analysis EDoS [39]	The execution trace analysis technique is used in different attack scenarios.	• It ignores important evaluation metrics such as cost and complexity. • A lack of a comprehensive analysis along with detailed attack scenarios and testbed, which raises doubt about its performance results.
11	Dynamic EDoS [17]	An unsupervised learning technique known as long short-term memory (a.k.a. LSTM) detects EDoS attacks using various metrics.	The performance evaluation of the scheme shows quite a low accuracy and consumes relatively high resource usage

From the above analyses and recommendations on defense inspired by two in-depth studies on EDoS characteristics proposed recently ([4] and [47]), we propose a novel effective solution to deal with EDoS attacks.

III. BACKGROUND KNOWLEDGE

A. EDoS ATTACKS ON SDN-BASED CLOUD

In EDoS attacks, by definition, the main purpose is to force a cloud user to rent extra computing resources by manipulating the auto-scaling engine on the cloud so that the user has to pay more money. As shown in Figure 2, before launching an EDoS attack, only three virtual machines are launched. A virtual machine (VM) is a virtual environment that functions as a virtual computer system. However, an EDoS attack exploits the auto-scaling engine in order to launch extra six VMs to meet the resource consumption of the cloud applications so that a cloud user is requested to pay for the attack activities.

Compared to a DDoS attack, an EDoS attack is stealthier and can be highly concealed [6]. A DDoS attacker aims to shut down the network system entirely by irrationally

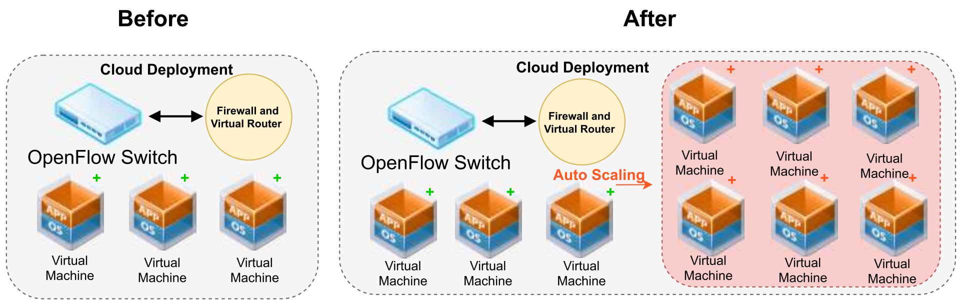

FIGURE 2. Before and after launching EDoS attacks.

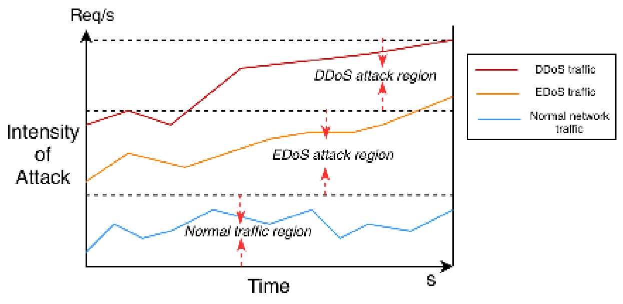

FIGURE 3. EDoS attack region with attack strength.

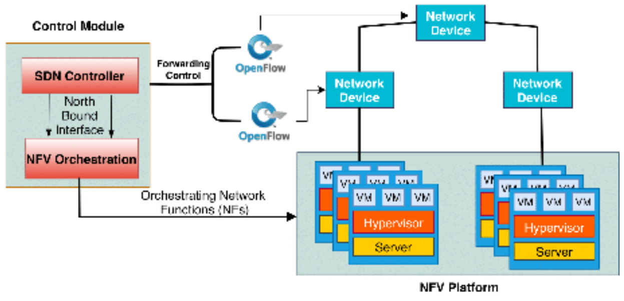

FIGURE 4. SDN-based cloud prototype.

by simply launching an attack with the utmost resources. Conversely, EDoS attackers are usually more sophisticated and highly logical. They increase the amount of unauthorized traffic gradually, and the main purpose is to force their target to pay more. EDoS attacks always try to stay below a traditional intrusion detection system's threshold value. In an SDN context, EDoS attacks not only degrades the SDN/NFV paradigm on the service provider end, but also forces a cloud user to pay for the unexpected attack event caused by an EDoS attacker. Figure 3 shows different traffic regions among normal traffic, EDoS, and DDoS. The intensity of EDoS attacks normally stays below the intensity of DDoS attacks. Therefore, an EDoS attack can easily bypass DDoS defense mechanisms. Thus, detecting an EDoS attack becomes a very complex challenge. An intelligent and robust approach is needed for cloud users and service providers to detect to detect EDoS attacks emerging in real time.

Compared to an EDoS attack in the cloud, an EDoS attack in an SDN-based cloud environment is basically similar. However, by integrating an SDN paradigm into a cloud environment, an EDoS attack can be easily detected because the SDN paradigm provides software-based traffic analysis [47], which means it provides various important traffic statistics provided by an SDN controller where it controls, manages, and collects all the network traffic. Therefore, we first propose a novel EDoS defense approach, leveraging the benefits of the SDN paradigm, to not only detect and mitigate different EDoS attacks effectively but also overcome the shortcomings of the existing approaches (as stated in sections I.A and I.C).

B. SDN-BASED CLOUD

SDN/NFV paradigm where two technologies NFV and SDN are combined (as showed in Figure 4) [2] includes a control module including SDN controller and an NFV orchestrator, and network devices. The SDN controller takes responsibility for managing the traffic path to exchange information with forwarding devices (OpenFlow switch) using the OpenFlow protocol primarily. And the NFV orchestrator is responsible for managing NFV services. VMs, which operate on top of hypervisors, are normally supported by hypervisors in launching and running network functions. Readers can refer to [2] for specific details of the SDN/NFV paradigm. Note that a VM in Figure 4 implies a virtual machine.

C. GRU, VAE, VARIATIONAL METHOD, AND NORMALIZING FLOWS

Recurrent neural networks (RNNs) [18] are a set of feed-forward neural networks commonly used for time series data. Unlike conventional neural networks, RNN's have an integrated internal memory to process sequences of inputs. However, RNNs are vulnerable to exploding and vanishing gradient problems. Moreover, they cannot process long sequences. Fortunately, a variant form of RNN called gated recurrent unit (GRU) can overcome RNNs' major disadvantages. It is explicitly designed to avoid the long-term dependency problem which normally arises in time series data [19]. Thus, in this work, we employ the GRU mechanism.

In detecting anomalies of higher-dimensional data like multivariate time series [21], a famous method known as a

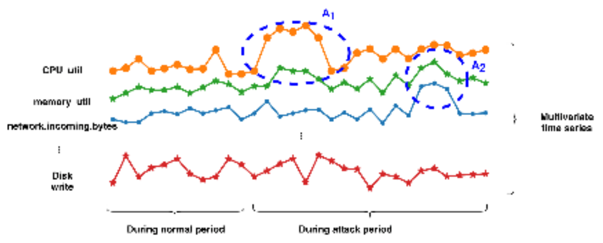

FIGURE 5. Two suspected regions highlighted in dashed-blue lines A_1 and A_2 occur during normal and attack periods.

variational autoencoder (a.k.a. VAE) [20] is often used. This provides a probabilistic model for describing an observations in latent space, and can also be transformed into a lower-dimension space preserving particular properties of the original dimension space.

Another variational algorithm known as stochastic gradient variational Bayes (SGVB) [20] commonly is used along with VAE to tune parameters. As proposed in [26], SGVB is not only used to tune parameters of both ϕ and θ but also maximize the evidence lower bound (a.k.a. ELBO).

To train a machine learning model, its parameters must be tuned for increasing the probability of predicting data in the training dataset under the model. However, this can be problematic if the output of the model is assumed to show a Gaussian distribution, as the true probability density function (PDF) of real data is often far from Gaussian. Therefore, Danilo Rezende *et al.* proposed a transformation technique called Planar normalizing flows (Planar NF) [22] to capture ($q_\phi(z_t|x_t)$) which is the non-Gaussian posterior density by transforming it using invertible mappings.

The basic idea of the combination of the four key techniques above is described as follows. GRU first captures complex spatiotemporal dependencies in state-space. Next, a variational algorithm (VAE) maps observations in state-space to stochastic variables. Next, a connection technique introduced in [27] known as linear Gaussian state-space modeling connects stochastic variables to GRU hidden variables [26]. Finally, a normalizing flow technique known as "planar flow" is used to support stochastic variables in *qnet* (defined in section IV.D.6) in capturing highly complex distributions of input data.

IV. R-EDoS: ROBUST EDoS DETECTION AND MITIGATION

In this section, our research rationale and an analysis of our system design are first given. Then, the components, workflow scheme, and the system process logic are introduced, respectively. Finally, the internal components of our proposed scheme are thoroughly explained. Also, all variables, constants, and parameters are summarized in Table 2.

A. PROPOSED SCHEME RATIONALE AND DESIGN ANALYSIS

Cloud users are normally monitored with various metrics over time when auto-scaling of resources is critical for service level agreements (SLA). For instance, as illustrated in Figure 5, cloud instances' parameters, such as CPU utilization (*cpu_util*), memory consumption (*memory_consumption*), number of incoming bytes (*network.incoming.bytes*) and so on, are collected and tracked by the auto-scaling engine. If an EDoS attack is initiated, some of the chosen metrics will abruptly change their values. Therefore, to detect anomalous periods in our multivariate time series during the attack, we need an intelligent model capable of learning from the network system. Thus, in this paper, we employ a method known as a stochastic recurrent neural network [26] that not only is able to learn from historical data but also to simultaneously keep track of the multivariate time series to detect an EDoS attack. The key idea of this technique is to output an anomaly score by learning from the patterns of the multivariate time series; then to compare this anomaly score with a dynamic threshold by using the reconstruction probabilities, thus detecting anomalies accurately.

A single time series involves consecutive observations that are appended at equally spaced timestamps [29]. Multivariate time series can be expressed as $x = \{x_1, x_2, \ldots, x_N\}$, where N indicates x's length, each observation (i.e. $x_t \in R^M$) is a vector with M dimensions, at time $t (t \leq N) : x_t = [x_1^{(t)}, x_2^{(t)}, \ldots, x_m^{(t)}]$, and $x \in R^{M \times N}$. The $x_{t-T:t}$ ($\in R^{M\times(T+1)}$) indicates a set of observations

TABLE 2. Notation definitions.

Notation	type	name
$p_\theta(z_t)$	variable	prior distributions for z_t
$p_\theta(x_t\|z_t)$	variable	posterior distribution for x_t
z	variable	latent representation
$q_\phi(z_t\|x_t)$	variable	an inference network
F	variable	a variance ratio
ϕ and θ	parameter	inference network parameters
Y_{ij}	variable	an observation
$\mathcal{L}(x_t)$	variable	ELBO
n	variable	a total # observations
T_i	variable	a group total
n_i	variable	a number belongs to group i
$X_{normalized}$	variable	a normalized array
G	variable	a grand total
x_{min}	variable	the smallest value in X
u*-s, **w***-s, and **b***-s	parameter	parameters of the corresponding layers
x_i	variable	a value of x in X
x_{max}	variable	the largest value in X
f^k	variable	invertible mapping functions
$\tilde{\mathcal{L}}$	variable	loss function
S_t	variable	anomaly score of x_t
$\overline{F}$	variable	a GDP function
γ and β	parameter	shape and scale parameters of GPD
th	variable	an initial threshold of anomaly scores
$\hat{\gamma}$ and $\hat{\beta}$	parameter	parameters to compute th_F using MLE
q	variable	a desired probability to observe $S < th$
th_F	variable	a final threshold
N'_{th}	variable	the number of S_i
q	constant	10^{-4}
low quantile	constant	$< 7\%$

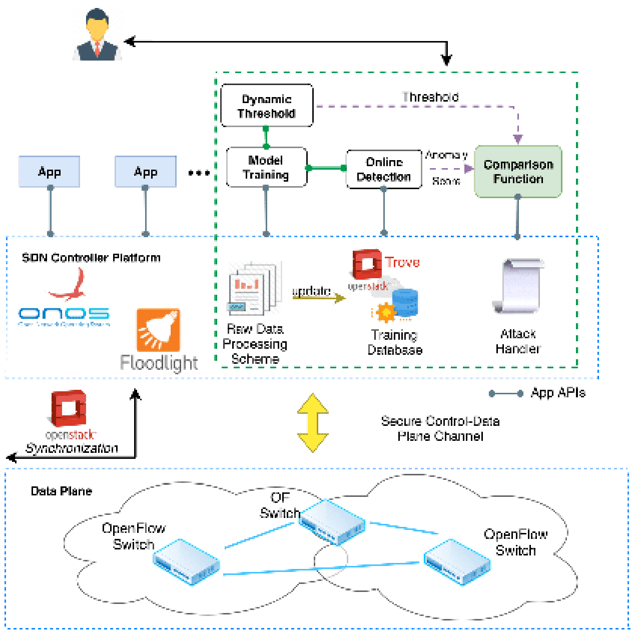

FIGURE 6. **Conceptual architecture of our EDoS defense scheme.**

from time $t-T$ to t written as $\{x_{t-T}, x_{t-T+1}, \ldots, x_t\}$. The objective of anomaly detection in multivariate time series is to decide if a new observation x_t is normal. Moreover, the historical data is crucial for predicting the current value of a sequence in a time series. Therefore, we take all the sequences $x_{t-T:y}$ to calculate an anomaly score. An anomaly score for x_t is produced from the anomaly detection model, then this anomaly score is compared against a dynamic threshold to calculate the anomaly prediction result.

Figure 6 is sketched to show a complete overview of our EDoS defense framework extending from both a control layer and a application layer with different modules. Specifically, a raw data preprocessing scheme, a training database, and an attack handler are placed in the SDN control layer; whereas a model training, a dynamic threshold, and an online detection modules are located in the SDN application layer.

B. R-EDoS MAIN MODULES AND SYSTEM WORKFLOW

Our EDoS defense scheme is placed in the SDN controller, including five main modules, as presented in Figure 7, which are a raw data processing scheme consisting of five sub-modules, an offline training model, an online detection, a dynamic threshold module, and an attack handler. This raw data processing module includes a traffic collector, a feature extractor, a transformation & standardization module, a Savitzky-Golay filter module, and an augmented Dickey-Fuller test, in that order.

The data preprocessing scheme is constructed from five different sub-modules and is also used by both model training and online detection. The process inside the module proceeds as follows. First, the traffic collector sub-module reliably gathers data on both data plane and instances at a certain periods of time. Subsequently, a feature extractor sub-module receives this data. Next some of the extracted features are selected as more relevant, and are further transformed into scaled data and smoothed using a Savitzky–Golay filter technique in order to raise the precision of the data without distorting the signal tendency and is checked stationarity using an augmented Dickey-Fuller sub-module. Once the data is identified as stationary, then it is divided into a series of sequences through sliding windows [30] ($T+1$ in length). After the raw data is preprocessed through this scheme, and given amount of time elapses (i.e. 5 minutes), the training module starts a training session using the preprocessed data. During training with the preprocessed data, the learning model tries to discern typical patterns in the data, and then scores a new observation. This scoring value is called an anomaly score, later compared with a dynamic threshold following the peaks over threshold (POT) approach [32].

Note that, to effectively adapt our proposed scheme to different network systems, we continually update the training database with new attributes that are collected from the raw data preprocessing in a preset time.

C. SYSTEM PROCESS LOGIC

As seen in Algorithm 1, N is represented a sequence of observations where each observation is an M-dimensional vector where each dimension indicates a network metric. Furthermore, $x^{\iota}_{t-T:t}$ $(\in R^{M\times(T+1)})$ is denoted as a set of observations $\{x^{\iota}_{t-T}, x^{\iota}_{t-T+1}, \ldots, x^{\iota}_{t}\}$ starting from time $t-T$ to t. The variable *curLength* denotes the length of current $x^{\iota}_{t-T:t}$. It is later compared with the length of $x^{\iota}_{t-T:t}$ at the next current time t. Before sending the sequence $x^{\iota}_{t-T:t}$ with T consecutive observations to both the offline model training ($\overline{OfflineMT}$) and online detection ($\overline{OnDetect}$) models, $x^{\iota}_{t-T:t}$ is fed into a transformation & standardization sub-module ($\overline{TSS}$) so that the input can be transformed, and standardized $x^{\iota}_{t-T:t}$. Once the input is already transformed and standardized in the $\overline{TSS}$ sub-module, a new sequence is formed from $x^{\iota'}_{t-T:t}$. This sequence is then filtered using the Savitzky–Golay technique ($\overline{SAGF}$) and is checked for stationarity or using an augmented Dickey-Fuller test ($\overline{AUDF}$). $x_{t-T:t}$ represents the sequence of successfully passing the stationarity test. Once an anomaly threshold th_F and an anomaly score $\overline{anomaly\_score}$ are obtained from the dynamic threshold module $\overline{D-thres}$, and $\overline{OnDetect}$; a new observation will determine whether the anomalous data indicates an attack as follows:

- If $S_{2t} >= th_F$ is true, then it indicates the new observation coming from a normal source. Thus, the algorithm is looped over again.
- Otherwise, the new observation is predicted to be derived from an attack source. The observation is sent to the attack handler to perform either *drop* or *remove* actions. Then, the algorithm is also looped over again.

D. INTERNAL MODULES

Here, we present all the components of R-EDoS in order, as shown in Figure 7.

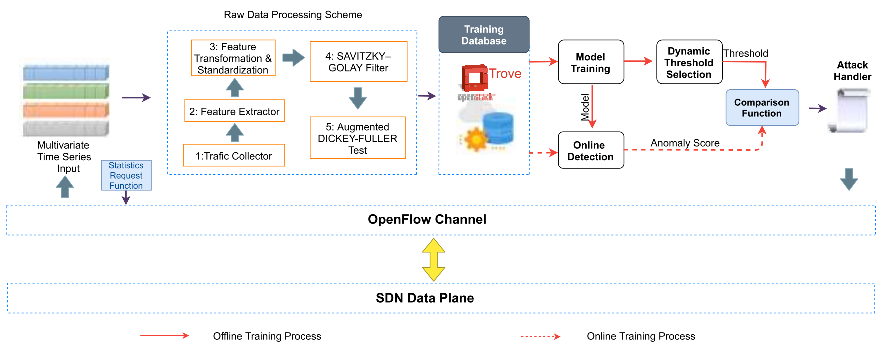

FIGURE 7. Main modules of our EDoS defense scheme sitting in the application layer of SDN.

Algorithm 1 R-EDoS: Robust EDoS Attack Defense Scheme

```
$N \longleftarrow$ A sequence of observations
$x^t = \{x^t_1, x^t_2, \ldots, x^t_N\} \longleftarrow$ A set of multivariate time-series including attributes of an observation
$M \longleftarrow M$-dimensions for each observation
$x^t_t = [x^{t1}_t, x^{t2}_t, \ldots, x^{tM}_t] \longrightarrow x^t_t \in R^M$
$\{x^t_{t-T}, x^t_{t-T+1}, \ldots, x^t_t\} \longrightarrow x^t_{t-T:t}$ $(\in R^{M\times(T+1)})$, from time $t-T$ to $t$
$\overline{TSS} \longleftarrow$ a transformation & standardization sub-module
$\overline{SAGF} \longleftarrow$ Savitzky–Golay filter sub-module
$\overline{AUDF} \longleftarrow$ augmented Dickey-Fuller test sub-module
$\overline{OfflineMT} \longleftarrow$ offline model training module
$\overline{OnDetect} \longleftarrow$ online detection module
$\overline{D-thres} \longleftarrow$ dynamic threshold module
$S_{1i}$ $(ano\_score) \longleftarrow$ output of $\overline{OfflineMT}$
$S_{2t}$ $(ano\_score) \longleftarrow$ output of $\overline{OnDetect}$ at time $t$
$curLength = (x^t_{t-T:t}).length()$
**loop**
  **while true**:
    **if** $(x^t_{t-T:t}).length() > curLength$ **do**
      $curLength += 1$:
      Send $x^t_{t-T:t}$ to $\overline{TSS} \longrightarrow x^{t'}_{t-T:t}$
      Send $x^{t'}_{t-T:t}$ to $\overline{SAGF} \longrightarrow x^{t''}_{t-T:t}$
      Send $x^{t''}_{t-T:t}$ to $\overline{AUDF} \longrightarrow x_{t-T:t}$
      Send $x_{t-T:t}$ to $\overline{OfflineMT} \longrightarrow S_{1i}$
      Send $S_{1i}$ to $\overline{D-thres} \longrightarrow th_F$
      Send $x_{t-T:t}$ to $\overline{OnDetect} \longrightarrow S_{2t}$
      **if** $S_{2t} < th_F$ **then**
        $normal\_source\_boolean = 0$ (attack traffic)
        Forward $x \longrightarrow AttackHandler$:
      **else**
        $normal\_source\_boolean = 1$ (normal traffic)
        **continue**
      **end if**
    **end if**
  **end while**
**end loop**
```

1) TRAFFIC COLLECTOR

There are two sources (data plane and instances) that the traffic collector needs to gather to guarantee the adequacy of all key features as described in Table 3. First, a statistics request function is run periodically to request statistics of an OpenFlow switch in order to collect data from the data plane (Figure 7). Once the OpenFlow responds to the request function, the OpenFlow channel forwards a *StatsResponse* message which includes requested statistics to the data processing scheme. Meanwhile, a Python script fetches other necessary statistics of cloud instances monitored by OpenStack Ceilometer.

TABLE 3. Top (17/38) key features.

Feature	Description
cpu_util	CPU utilization
memory_consumption	memory consumption
disk_usage	disk usage
no_flows	# flows
network.incoming.packets	# incoming packets
network.outgoing.packets	# outgoing packets
protocol_type	network protocol
no_pkt_flow	# packets per flow
no_bytes_flow	# bytes per flow
src_ip	source IP
dst_ip	destination IP
flow_duration	duration of a flow
no_established_connections	# established connections
network.incoming.bytes	# incoming bytes
network.outgoing.bytes	# outgoing bytes
no_syn_pkt	# SYN packets
no_ack_pkt	# ACK packets

2) FEATURE EXTRACTOR

At this stage, we run a feature selection process to automatically select features that contribute the most to the classification decision, applying the Boruta algorithm [34] to select both unbiased and stable attributes. There are some factors that make Boruta a notable success in finding top important features from particular data like time series; for instance, it takes both the interactions and the relationships of multiple variables into account. The Boruta algorithm workflow can be summarized as follows. First, it duplicates the original dataset; then the *z_score* values, which are acquired from the two dataset using a random forest algorithm, are used to compare with each other. Next, if *z_score* value in the

Algorithm 2 R-EDoS: Feature Extractor Module - *Boruta*

```
in_param feature set: $F_S = \{f_1, f_2, \ldots, f_t\}$, dataset: $D$
out_params a set of selected features $F_{select} = attributes[]$
Read $D$
$no_f s \longleftarrow attributes[].length$
for $f = 1$ to $no_f s$ do:
    Create shadow variable $var_f^{shadow}$
    Embed $var_f^{shadow}$ to the information system
end for
$var_{shuf}^{shadow} \longleftarrow$ Randomly shuffle the $var_f^{shadow}$ across objects.
$z_{score}[] = \textbf{RandF\_Clf}\ (train\_set, var_{shuf}^{shadow})$
Define $max = 0, r = 0$
while $r \leq max\_RandF\_runs$ do:
    for $f = 1$ to $num_f$ do:
        $z_{score}[] = \textbf{RandF\_Clf}\ (train\_set, attributes[f])$
        $max_{z\_score} \longleftarrow z\_score[f]$
        if $z\_score[f] > max_{z\_score}$ then:
            $attributes[f] \longrightarrow$ "Im"
        else if $z\_score[f] < max_{z\_score}$ then:
            $attributes[f] \longrightarrow$ "Un"
        else:
            Perform two sided equality test
            $(\forall attributes[f] \neq$ "Im") && $(\forall attributes[f] \neq$ "Un")
            $attributes[f] \longrightarrow$ "Und"
        end if
    end for
end while
return $attributes[f]$
Note that: "Und": Undetermined, "Im": Important, "Un": Unimportant
```

Algorithm 3 Define *RandF_Clf* Function

```
def RandF_Clf(train_set, feature):
{
    Define $T \longrightarrow$ # trees
    $counter = 0$
    for $1 = 1$ to $T$ do:
        Bootstrap samples with $m$ variables
            ∀m, calculate the importance $var_{ip}$ based on gini impurity measurement
        $DST_t \longrightarrow$ decision tree of $m$ variables
        Define $node_{min} \longrightarrow$ minimum node size
        if $counter \leq node_{min}$ then:
            Add node to $DST_t$
            $counter += 1$
        end if:
        $node_{D_t} \longleftarrow$ # nodes in $DST_t$
        $node_{ip} = \frac{\Sigma_{n=1}^{node_{D_t}} var_{ip}}{node_{D_t}}$
        Calculate variance $\sigma_{node}$ in $D_t$
    end for
    for all $T$ do:
        Calculate $z\_score(D_t) = \frac{node_{ip}}{\sigma_{node}} \times \sqrt{node_{D_t}}$
        $z\_score[] \longleftarrow z\_score(D_t)$
    end for
    return z_score[]
}
```

duplicated dataset is consecutively smaller than in the original dataset, then the selected feature is chosen.

The following Algorithm 2 and 3 explain the processes of feature selection with Boruta in detail. First, an addition system is created by using shadow samples (var_f^{shadow}) for all variables $f_t \in F_S$ in which all the features are mixed to eliminate biases and high correlations among them. Next, a random forest algorithm (*RandF_Clf*) is trained using this addition system, and then it is used to evaluate the importance level of each feature (var_{ip}). Classification is then conducted using decision trees from different bagging samples. The *z_score* score is then calculated. It is noted whether the *z_score* score is statistically significant for the feature; if not the procedure is run again several times. During every iteration, by comparing a random set of attributes with actual attributes, a maximal importance (max_{z_score}) is calculated. For all the attributes, a statistical test is conducted. After the test, trivial attributes are removed. If the importance value of an attribute is greater than max_{z_score}, then the attribute is considered "important". On the contrary, If the importance value of an attribute is considerably smaller than max_{z_score}, then the attribute is considered as "unimportant" and will be removed. Lastly, attributes that are marked as "undetermined" receive no further actions.

3) FEATURE TRANSFORMATION & STANDARDIZATION

This module is responsible for transforming all the selected features from the previous stage. First, the categorical features such as *src_ip* and *dst_ip* are transformed into numeric variables using one-hot encoding [35]. Second, as stated in section I.A, unlike existing solutions which are limited to some specific distributions (typically a Poisson distribution), in this paper, we employ a method called Yeo-Johnson transformation [16] which is a power transform. Yeo-Johnson not only stabilizes variance, which can solve issues related to heteroscedasticity, but also normalizes the data distribution, transforming it to more closely resemble a Gaussian distribution, to increase the prediction accuracy of our detection model without having to take into account the mixed distributions of the attack traffic. Therefore, all the selected features are further transformed based on Yeo-Johnson's formula as follows.

$$d_i^{(\beta)} = \begin{cases} ((d_i+1)^{\beta}-1)/\beta, & if\,\beta \neq 0, d_i \geq 0 \\ log(d_i+1), & if\,\beta = 0, d_i \geq 0 \\ -[(-d_i+1)^{(2-\beta)}-1]/(2-\beta), & if\,\beta \neq 2, d_i < 0 \\ -log(-d_i+1), & if\,\beta = 2, d_i < 0 \end{cases} \quad (1)$$

TABLE 4. ADF test result.

Type	Value
ADF	-56.02
p value	< 0.0001
Rejection degree of 1%	-3.430
Rejection degree of 5%	-2.862
Rejection degree of 10%	-2.567

In equation 1, d_i indicates the preprocessed data, and β indicates the transformation parameter. Note that we also set $standardize = True$ to standardize the preprocessed data.

4) SAVITZKY–GOLAY FILTER

After being transformed and standardized in the previous step, the data is further smoothed to remove noise and outliers caused by non-stationary time series. Specifically, some data may be lost or unusual while collecting or transmitting data, and this may cause interference data. To smooth the original sequence, remove noise, and retain time series' peak and width, the least square polynomial smoothing method (also known as a Savitzky–Golay (SG) filter) [36] is used. The method works best with a window size of 5.

A time series is represented as:

$$X = \{x_1, x_2, \ldots, x_t\}, t \in N^+ \tag{2}$$

where x_t means a set of tasks at time slot t, $N^+ = \{1, 2, \ldots\}$, X is the workload, and Y_n ($n[m+1, t\text{-}m]$) indicates a subsequence of X. Y_n is defined as:

$$Y_n = \{x_{n-m}, \ldots, x_n, \ldots, x_{n+m}\}, n \in [m+1, t-m] \tag{3}$$

To find the best mean square, a set of $2m+1$ consecutive values is used. In which coefficients of a polynomial are as follow:

$$p(n) = \sum_{r=0}^{\Upsilon} a_r n^r, n \in [-m, m] \tag{4}$$

It is noted that the value of n ranges from $-m$ to m, and $n = 0$; therefore, the least-square criterion requires that the squares of the differences in total between the observed values x_{m+n} and the computed values $p(n)$ be the minimum over the time slot being observed [37], defined as:

$$\varepsilon = \sum_{n=-m}^{m} (p(n) - x_{m+n})^2 \tag{5}$$

Then, the central point of the fitted polynomial implies the smoothed data point.

5) AUGMENTED DICKEY-FULLER TEST

For the model to learn from time series data reliably, it is crucial to check whether a sequence is stationary or not, therefore, a test called augmented Dickey–Fuller (ADF) [37], also known as a unit root test, is employed to check the stationarity of each data sequence. The role of ADF is to determine how strongly a time series is characterized by a trend. The test checks if a unit root is included in a sequence; if not then the sequence is non-stationary; otherwise, it is stationary.

For the ADF experiment, the null hypothesis ($H0$) occurs if the time series is considered as non-stationary. Therefore, if the processed sequence fails in the stationarity test, a differencing technique is run, after which the sequence will be tested again. On the contrary, if $H0$ is rejected, it suggests that the examining sequence does not have a unit root, meaning that it is stationary; therefore, it is then sent to the training model.

In ADF, there are three degrees of rejection 1%, 5%, and 10%, in which a rejection degree of 1% indicates strict rejection of the original hypothesis. That means if the ADF value is less than the rejection degree of 1%, then the examining sequence is not non-stationary. Moreover, we interpret the test result using the *p-value*. If the *p-value* lies under the rejection degree of 5% or 1%, then it indicates stationarity (rejection of $H0$), otherwise a *p-value* lies above the rejection degree, indicating nonstationarity. For example, the results of the ADF test of a *flow_duration* feature are shown in Table 4. The calculated ADF value is much smaller than the rejection degree of 1%, and the *p-value* is extremely low (< 0.0001). That means the selected feature is stationary and is qualified to be used in training the model.

6) MODEL TRAINING

The model training is composed of two main networks: *qnet* and *pnet* (Figure 8(a) and 8(b), respectively). First, in the *pnet* network, $z_{t-T:t}$, indicating a latent representation, is used to reconstruct $x_{t-T:t}$. The more precise the representation of $x_{t-T:t}$, the lower the reconstruction loss. Second, the *qnet* network approximates the *pnet* network by optimizing itself in order to obtain a higher quality latent representation.

As proposed in [26], by tuning three network parameters ($\mathbf{u}$*-s, $\mathbf{w}$*-s, and $\mathbf{b}$*-s), both the *qnet* network and *pnet* network are trained at the same time. The size of each input sequence data (e.g., $x_{t-T:t}$) is $T+1$. According to [26], for the *l-th* sample $z^{(l)}_{t-T:t}$, where $q \leq l \leq L$ and L is the sample length, the loss function is formulated as:

$$\begin{aligned}\widetilde{\mathcal{L}}(x_{t-T:t}) \approx \frac{1}{L}\Sigma_{l=1}^{L}[&log(p_\theta(x_{t-T:t}|z^{(l)}_{t-T:t}) \quad [1]\\ &+log(p_\theta(z^{(l)}_{t-T:t})) \quad [2]\\ &-log(q_\theta(z^{(l)}_{t-T:t}|x_{t-T:t}))] \quad [3]\end{aligned} \tag{6}$$

In Equation 6 given above, for each sample:

- The first term is the negative reconstruction error and it is equal to $\Sigma_{i=t-T}^{t} log(p_\theta(x_i|z_{t-T:i}))$. The posterior probability of x_i is formulated as: $p_\theta(x_i|z_{t-T:i}) \sim \mathcal{N}(\mu_{x_i}, \sigma^2_{x_i} I)$ [26].
- The second term is equal to $\Sigma_{i=t-T}^{t} log(p_\theta(z_i|z_{i-1}))$, in which z_i is acquired by linear Gaussian state space modeling proposed in [26].
- The third term approximates the authentic posterior distribution in the $z-$space of the *qnet* of z_i and it is equal

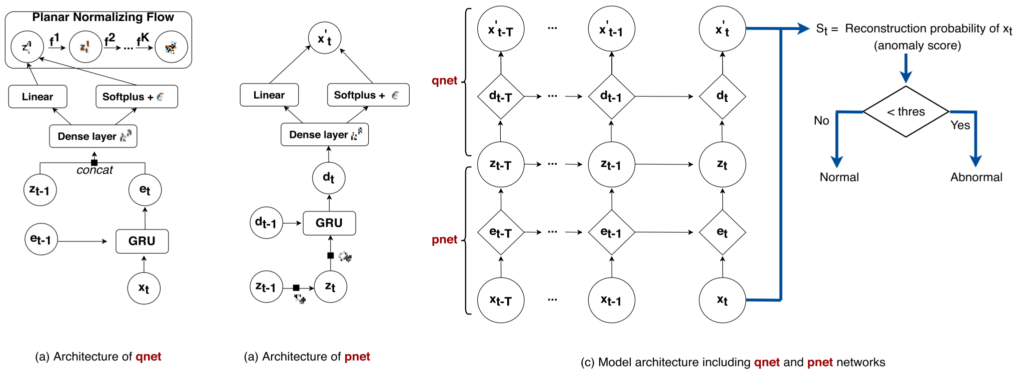

FIGURE 8. Model architecture.

to $-\Sigma_{i=t-T}^{t} log(q_\theta(z_i|z_{z_i-1}, xt - T : t)$ where z_i (i.e., z_i^k) is converted from a normalizing flow technique called planar NF (section III.C). $Z_i^K = f^K(F^{K-1}(\ldots f^1(z_i^0)))$, in which $z_i^0 = \mu_{z_i} + \xi_i \sigma_{z_i}$, $\xi_i \sim \mathcal{N}(0, I)$, and f^k is invertible function mapping [26].

7) ONLINE DETECTION

At this stage, the trained model is used to determine whether or not an observation at some given time step (i.e., x_t) is anomalous. It is noted that the input of our learning model is a set of sequence data with length $T + 1$. Hence, $x_{t-T:t}$, including a sequence of observations, is taken as an input to reconstruct x_t [25]. This reconstruction then acts as an anomaly score by applying the conditional probability, as suggested in [23]. S_2t represents the anomaly score of x_t, so $S_2t = log(p_\theta(x_t|z_{t-T:t}))$ [26]. A high value of S_2t implies x_t is reconstructed well, meaning that the observation closely follows the usual patterns of the time series. In conclusion, if the calculated threshold value is greater than S_{2t}, then the observation x_t is abnormal and vice versa.

8) DYNAMIC THRESHOLD MODULE

First, an anomaly score is calculated while running the model training module, in which each observation is represented as a multivariate time series of N'. The module then produces a univariate time sequence in the form of $S_1, S_2, \ldots, S_N$, as illustrated in Figure 7. Second, an anomaly threshold (th_F) is calculated by applying extreme value analysis (EVA) [31]. EVA is a statistical method that is commonly applied to discover extreme deviations of a probability distribution without making assumptions about the distribution. According to [31], the extreme values are often located in the tail of the probability distributions. A common approach in EVA is generally referred to as peaks-over-threshold method (POT) [32], which learns the tail area of the probability distribution using a generalized Pareto distribution (GPD) with two parameters, which must be tuned ($low\_quantile < 7\%$ and $q = 10^{-4}$). Readers can refer to [26] for further details. Formally, anomaly threshold is defined as follows.

$$th_F \simeq th - \frac{\hat{\beta}}{\hat{\gamma}}((\frac{qN'}{N'_{th}})^{\hat{\gamma}} - 1) \tag{7}$$

In Equation 7, N' is the number of observations, N'_{th} the number of S_i s.t. $S_i < th$, and q the intended probability of finding that $S < th$.

9) COMPARISON FUNCTION

This function simply compares the calculated threshold to the calculated anomaly score. If $S_{2t} < th_F$ is true, the model determines that the input data indicates attack traffic, and the attack traffic is sent to an attack handler. Otherwise, it was determined to be a normal traffic.

10) ATTACK HANDLER

Once the training model of R-EDoS detects attack traffic, then an attack handler forwards a *flow_mod* message, including a *delete* action, to the OpenFlow switch. The attack handler requests the SDN controller to perform dropping *packet_in* messages of the attack traffic (also presented in Figure 7).

V. EXPERIMENTAL SETUP

In this section, we show common EDoS attack scenarios that are also simulated by us in this work. Next, the description of our dataset is given. We also show readers our sensitivity analysis and a comprehensive description of model training. Finally, we describe our implementation in an SDN-based cloud.

A. ATTACK SCENARIOS

There are no existing works introducing comprehensively what are some common EDoS attack scenarios in an SDN-based cloud that are normally used by an attacker. In this work, we divide into five common EDoS attacks that are mentioned in these studies ([12], [17], [39] and [42]). Note that we also followed their instructions to launch these five different EDoS attacks.

1) TCP—HTTP FLOODING ATTACK (HTTPAttack)

An HTTP flood attack [39] is one of the DDoS attacks where a webserver is exploited by an attacker through HTTP GET or POST requests. The attacker forces the targeted webserver to allocate the utmost resources for each request. The request is normally a resource-intensive request.

2) DATABASE API REQUESTS (APIRequests)

In the case of Database API requests attack, a virtual machine works as a database server where it is frequently queried through HTTP requests. A cloud provider usually charges a user for the number of API requests. For example, a cloud user wants to create a custom dashboard that makes calls to a relational database RDS provided by AWS [48]. The user needs two APIs such as *DescribeDimensionKeys*, and *GetResourceMetrics* and the dashboard calls these two APIs every 5 seconds. The number of calls in 30 days would be: 2 API calls * (30d * 24h * 60m * 60s) / (5s) = 1,036,800 API calls. If AWS provider monitored only two instances with the custom dashboard refreshing every 5 seconds, the total cost would be: ((2 * 1,036,800 - 1,000,000) / 1,000) * 0.01 = \$10.74 per month. By manipulating this pricing calculation, an EDoS attacker can create more fake API calls more frequently and the user needs to pay extra fees.

3) TCP—SYN FLOODING ATTACK (SYNFlooding)

An SYN flooding attack [39] also known as "half-open attack" aims at consuming all available resources of a server so that it is unable to respond to upcoming traffic. This attack behavior is similar to DDoS attacks. An attacker continuously sends an initial connection request to a targeted server making all ports unavailable to respond to upcoming legitimate traffic.

4) YO-YO ATTACK (YoYo)

Attacking the auto-scaling engine literally means EDoS attack, in which Yo-Yo attack [42] with two modes (on-attack and off-attack) is revealed attack against the auto-scaling engine. In the first mode, an EDoS attack forwards a burst of network traffic to trigger the scaling engine so that more instances are launched. Conversely, in the second mode, after the attacker believes that all instances are up and the service is functional; then it will halt forwarding network traffic until it verifies that scale down has occurred.

5) SLOWLORIS ATTACK (Slowloris)

Slowloris attack [46] is commonly known as "low and slow" attacks because it utilizes low bandwidth and aims at mimicking normal traffic. Its goal is to open and to keep many connections as long as possible by using partial HTTP requests and targeting a webserver on the cloud. Once the attack is launched, the web server will be overloaded because there are so many opening threads that need to handle. If the connections exceed the web server's maximum connections, then a denial-of-service phenomenon will emerge, which means upcoming requests will not be responded to.

TABLE 5. EDoS attack simulation with different number of requests.

Scenario	Number of Requests
S-1	1000
S-2	2000
S-3	3000
S-4	4000
S-5	5000
S-6	6000
S-7	7000

Note that EDoS attacks are similar to low-rate DDoS in characteristics [4]. For example, as specified in [3], a high-rate DDoS attack occurs when the packets per second $> 10,000$; therefore, 10,000 packets/s are considered the standard threshold to differentiate between low rate DDoS and high rate DDoS. Moreover, low rate DoS attacks normally account for 10- 40% of normal traffic [3]. Hence, based on both the studies, to create realistic EDoS attacks with the five attack scenarios introduced above, we stimulated different levels of EDoS attacks, in which EDoS attacks always account for 20% of normal traffic, and the number of requests ranges from 1000 to 7000 following the EDoS evaluation scheme proposed by Al-Haidari *et al.* [43] as shown in Table 5.

B. DATASET DESCRIPTION

One of the most realistic data of network traffic with various metrics such as *CPU usage, network, and memory consumption, and so on* of different servers known as SMD was chosen. This new dataset was publicly published in KDD 2019 [26] and was collected from real network traffic statistics of a large Internet company. It is also used in many anomaly detection studies mentioned in [25]. In our study, the offline model module was trained and tuned its hyper-parameters using the SMD dataset, in which we sampled this dataset with 20% attack traffic. The shapes of the training set and testing set are (28479, 38) and (20300, 38), respectively. Furthermore, to evaluate our proposed scheme with a real-life dataset, we also collected more data samples from our simulated testbed environment with the shape of (16250,38) including attack and normal traffic.

C. SENSITIVITY ANALYSIS OF USER-DEFINED PARAMETERS WITH TAGUCHI AND MODEL SETUP

To achieve the best results of R-EDoS where the number of experiments is as little as possible, it is essential to investigate its user-defined parameters. In our proposal, four highly sensitive parameters are needed to find their best values, which are data sequence length, $z-$ space), planar NF length, and GRU layers and dense layers. In order to pick a fair combination between factors rather than the entire combination, we use the popular Taguchi method proposed in [28], in which fractional factorial designs (a.k.a. OAs),

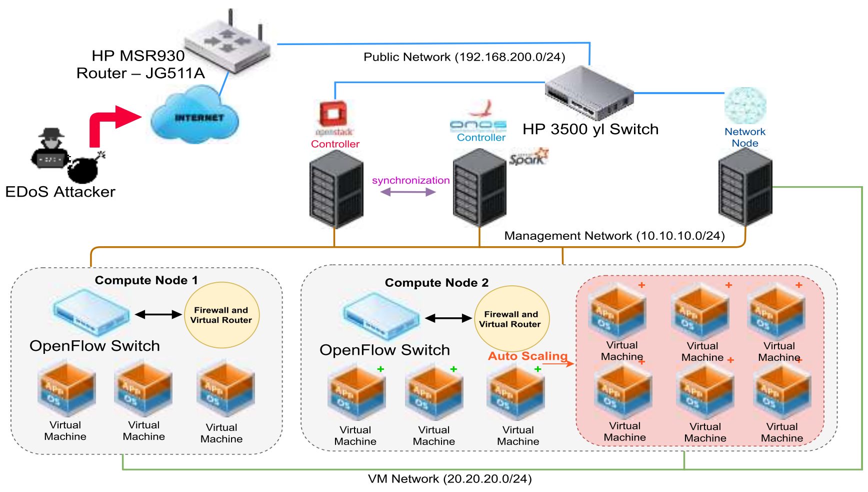

FIGURE 9. Our SDN-based EDoS testbed deployed on OpenStack cloud platform.

TABLE 6. Cluster settings in detail.

No. of Machines	Node type	Processor	Memory	Disk
1	OpenStack Controller	Intel®i7-8700 (12) @ 4.6000GHZ	64237MiB	SSD 512GB ATA
1	ONOS Controller + Spark Master	Intel®i7-8700 (12) @ 4.6000GHZ	47936MiB	SSD 512GB ATA
1	Network Node	Intel®i7-4770 (8) @ 3.900GHz	15930MiB	SSD 128GB ATA
1	Compute-1 + Spark Slave-1	Intel®i3-6100 (4) @ 3.700GHz	7672MiB	SSD 240GB ATA
1	Compute-2 + Spark Slave-2	Intel®i7-4770 (8) @ 3.900GHz	24004MiB	SSD 240GB ATA

TABLE 7. Model parameters.

data sequence length	100
gradient_clip_norm	10.0
initial_learning_rate	0.001
GRU & dense layers	500 units
window_length	100
ϵ	10^{-4}
z-space dimension	3
epochs	20 (early_stopping=true)
optimizer	Adam
gradient clipping	10.0
batch_size	50
dense_dim	500
L2 regularization	10^{-4} coefficient
nf_layers	20
std_epsilon	0.0001
test_batch_size	50
valid_step_freq	100
q	10^{-4}
x_dim	38
POT parameters (3 subsets)	(0.8908, 0.8032, 0.9999)
boruta depth tree	7

where relies on both their levels and the number of control factors, are used to find the number of experiments. Multiple process variables can be estimated by using OA design while reducing the number of test runs. In our work, four key factors and their levels are: *data sequence length* $\in \{50, 100, 150, 200, 250\}$; *(z-space)* $\in \{2, 3, 4, 5, 6\}$; *planar NF length* $\in \{5, 10, 15, 20, 25\}$; *GRU and dense layers* $\in \{100, 200, 300, 400, 500\}$. If we run a grid search method, a full factorial design would require a total of 5^4 or 625 possible combinations, which is an enormous number of experiments to run. Fortunately, based on the Taguchi method, the total required experiments are only 25 with an orthogonal array of $L_{25}(5^4)$.

To produce the best outcomes and get valuable insights significantly from the learning model of R-EDOS, we obtained the best settings of user-defined parameters based on the Taguchi results, and we also closely followed the descriptions in [26]. Based on the sensitivity analysis Taguchi given above, the **z**-space dimension equaled 3 and the planar NF length was assigned as 20. Besides, the GRU and dense layers are equal to 500 units and the ϵ was set to 10^{-4} in the standard deviation layer. The input data sequence is 100 in length. Also, here are other hyper-parameters settings. The model was trained with 20 epochs and a batch size of 50 including the early stopping regularization criterion. Adam optimizer was selected as an optimization algorithm for stochastic gradient descent. Besides, the value of the learning rate was 10^{-3}. While running back-propagation, gradients may increase their values rapidly with a very large number resulting to the overflow problem (i.becomes *float.nan*) of model parameters [26]. To avoid this problem, the gradient clipping was used and was set to 10. In addition, all layers in the learning model were penalized by applying the L2 regularization technique. Finally, the hyper-parameters settings used in this work are summarized in Table 7.

In Boruta, an irregular *z_score* value problem may cause an infinite loop issue. Fortunately, the issue can be addressed

TABLE 8. Anomaly detection performance comparison on average between R-EDoS and other solutions in detail.

	Detection Rate (%)	Accuracy (%)	$\overline{FAR}$ (%)	F1-score (%)
R-EDoS	96.27	94.33	4.72	95.51
Our Previous Work [17]	91.4	89.86	9.75	90.88
NN - Abbasi *et al.* [39]	77.35	79.41	20.86	76.58
SVM - Abbasi *et al.* [39]	85.08	85.92	15.61	85.4
One-Class SVM [40]	81.5	83.17	17.22	82.2

by setting the depth to 7 [33]. Hence, in our work, we also set the depth of tree = 7.

D. DETAILED IMPLEMENTATION AND TEST PREPARATION

In this work, as suggested in [44], we set up our OpenStack-SDN based testbed according to it. Figure 9 displays the testbed in a production environment for virtual networks to simulate different EDoS attacks. Our testbed consists of an OpenStack platform (one controller of SDN known as (ONOS Quail version 2.0.0) [49], a controller of OpenStack, two computing nodes attached with OvS version 2.3.2), and one networking node). In which, Apache Spark is also installed into our cluster so that we can not only leverage an optimal preprocessing process but also the learning process for machine learning algorithms. We used OpenFlow version 1.4 in this work. Bonesi simulator was employed to perform flooding attacks with low attack rates, besides sFlow and sFlow-RT tools were employed to perform collecting data from the SDN paradigm. As we introduced in section IV-A, the modules of R-EDoS are sat in the ONOS node (Figure 6). Our system configurations are presented in Table 6 running on Ubuntu operating system 64bit version 18.04. It is noted that we created an auto-scaling group on OpenStack with nova computing instance configurations as follows, instance type: m1.small, VCPUS: 1, memory: 2048MB, storage: 20GB. For two particular attacks (HTTPAttack and Slowloris), we created an auto-scaling group in which instances work as a web server. And for an APIRequests attack, OpenStack service named Trove is a database as a service of each instance.

To run both Savitzky–Golay filter and augmented Dickey–Fuller test, *SciPy* package and *statsmodels* were used respectively. Also, the Yeo-Johnson was run by using *sklearn* package and chose its λ parameter automatically.

VI. RESULT ANALYSIS

In this section, we first compare R-EDoS with other solutions and then present detection performance. Next, resource usage and the overall performance of our R-EDoS scheme are analyzed. Then, we interpret the learning model of R-EDoS and then briefly discuss our deployment. We conclude the section with a general discussion.

A. COMPARISON WITH OTHER SOLUTIONS

To prove the effectiveness of R-EDoS, we conducted practical experiments to make comparisons with other existing works using the same environmental set-up. First, to the best of our knowledge based on a review of the relevant literature. Abbasi *et al.* [39] is the only work applying machine learning technique to an EDoS detection problem, excluding our previous work [17]. Hence, we first compared our proposed scheme with their two approaches (support vector machine (SVM) and neural network (NN)) independently). Second, to prove our approach, which is multivariate time series detection, to be practical, we also compared it with another machine-learning-based method called one-class SVM [40]. According to Schölkopf *et al.* [40] one-class SVMs separate all the data points into two spaces where the distance between the two spaces are maximized as much as possible. It is a well-known algorithm and has been used in many anomaly detection studies [41]. Last, our previously published work [17], was also selected for comparison to demonstrate the expected enhancement of R-EDoS.

It is noted that we implemented these studies, [17], [39], and [40], in our testbed to keep not only their novel features but also their original functionalities.

B. ANOMALY DETECTION PERFORMANCE

To rigorously assess the anomaly-detecting capacity of R-EDoS, four criteria, including F1-score, detection rate, accuracy, and false alarm rate mentioned in [45] are employed. To calculate the four criteria, we need four other different measurements including false positive (FP), false negative (FN), true positive (TP), and true negative (TN). TP indicates the ratio of illegitimate traffic that is recognized as such TN means the percentage of normal traffic that is detected as anomalous traffic, FP shows the percentage of legitimate traffic that is detected as anomalous, and FN reflects the percent ratio of abnormal traffic that is classified as normal traffic. The formula of the four criteria are expressed as follows:

- Detection Rate ($\overline{R}$) is the proportion of the number of detected abnormal flows to the number of all abnormal flows:

$$\overline{R} = \frac{TP}{TP + FN} \tag{8}$$

- Accuracy ($\overline{AC}$) is the proportion of correct detection over the total the number of total flows:

$$\overline{AC} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

- False Alarm Rate ($\overline{FAR}$) is the ratio of anomalous flows falsely identified as legitimate flows:

$$\overline{FAR} = \frac{FP}{FP + TN} \tag{10}$$

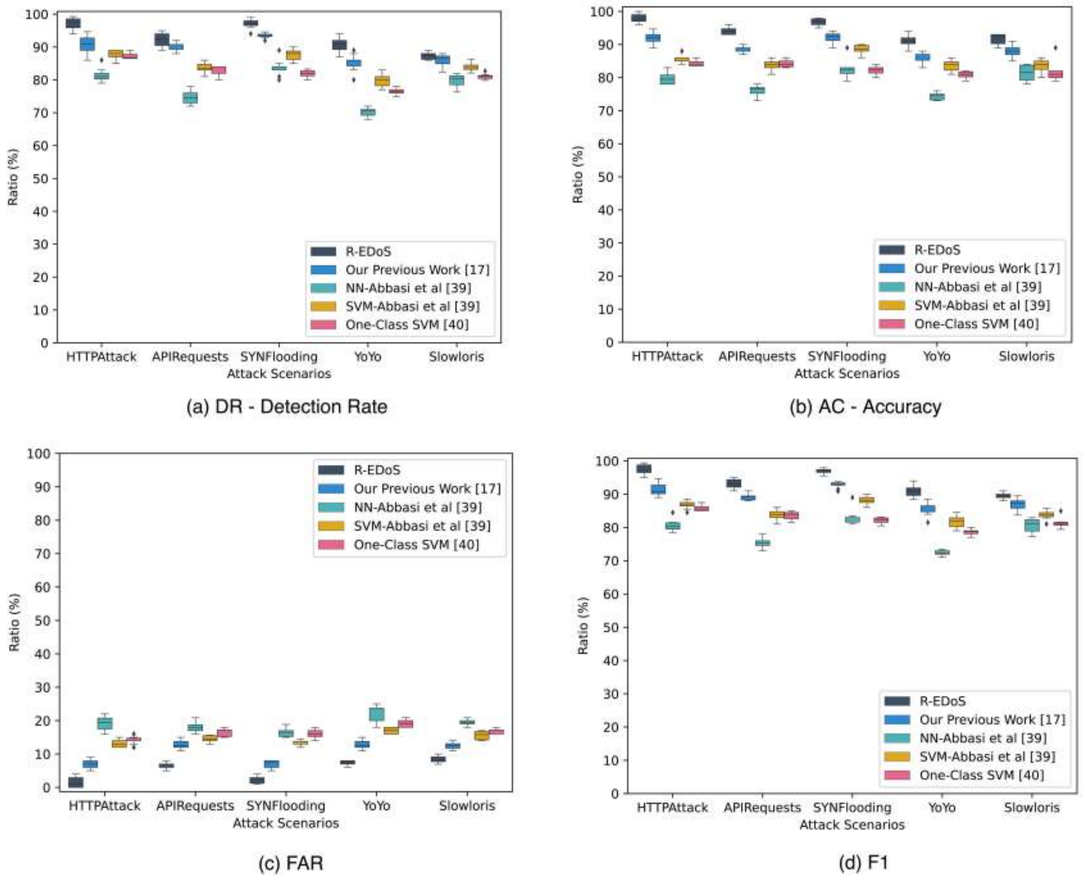

FIGURE 10. Detection performance with 4 distinct measurements in different kinds of EDoS attacks.

Hence, this score takes both FPs and FNs into account.:

- F1-score (F1) is the weighted average of $\overline{P}$ and $\overline{R}$.

$$F1 = \frac{2 * \overline{P} * \overline{R}}{\overline{P} + \overline{R}} \tag{11}$$

where $\overline{P} = \frac{TP}{TP+FP}$

In summary, the detection rate reveals the percentage of correctly detected attacks over the total number of attacks. $\overline{FAR}$ represents the ratio of falsely detected attacks to the total of falsely identified entities, whereas accuracy tells how accurate R-EDoS is. Lastly, F1-score takes both FPs and FNs into account implying the weighted average of $\overline{P}$ and $\overline{R}$.

Figure 10 shows the detection performance of five mentioned solutions: R-EDoS, NN-Abbasi *et al.* [39], SVM-Abbasi *et al.* [39], our previous work [17], and one-class SVM [40]. The results were acquired through testing on 10 sub test sets. It is obvious that R-EDoS both outperforms the other solutions in regards to accuracy, F1-score, and detection rate and produces the lowest false-alarm rates under different EDoS attack scenarios. Conversely, the NN-Abbasi *et al.* solution performs poorly regarding to the accuracy, detection rate, and F1-score. Furthermore, it produces false-alarm rates with the highest rate. Among the three other solutions, it is shown that our previous work still performs better than both SVM-Abbasi *et al.* and one-class SVM solutions, while the former performs slightly better than the latter.

Based on formulas [9]−[12], we calculated anomaly detection performance metrics on average as presented in Table 8. Regarding accuracy and detection rate, R-EDoS and our previous work clearly outperform the other three solutions. R-EDoS demonstrates stable prediction performance and it always achieves the highest scores among the models compared, with a detection rate of 96.27%, and an accuracy of 94.33%. Moreover, our two works also produce fewer wrong warnings than the others, with R-EDoS only showing 4.72% of the total test traffic sequences as false-positives, whereas [17] had a higher false-positive rate, 9.75%. In comparison among NN-Abbasi *et al.* [39], SVM-Abbasi *et al.* [39], and one-class SVM [40], a pure neural network struggles to learn from higher dimensional time series data; therefore, it performs poorly in detection and produces a very high false-alarm rate, 20.86%. One-class SVM is specifically designed for detecting anomalies and its main advantages are less time and storage space to run compared to two-class SVM [41]; therefore, as shown in Table 8, its prediction performance is slightly poorer than SVM-Abbasi. Lastly, the F1-score is an appropriate measurement criterion to properly assess the efficiency of R-EDoS since it is taken

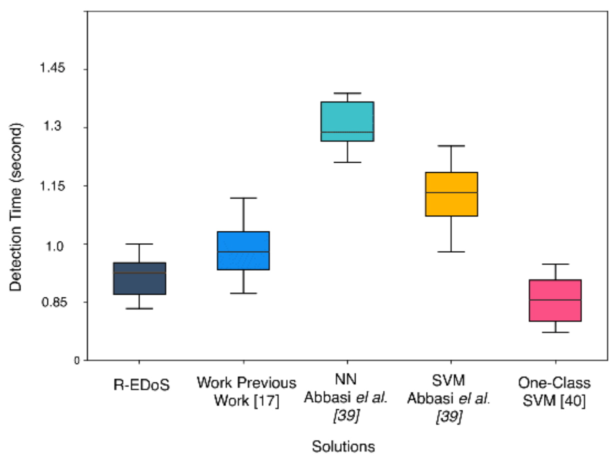

FIGURE 11. Detection time among different solutions.

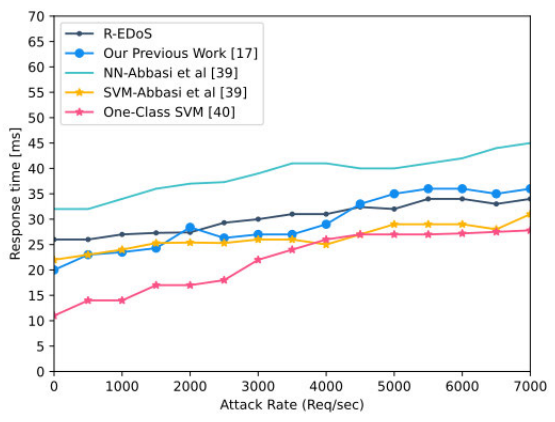

FIGURE 12. Response time according to attack rate.

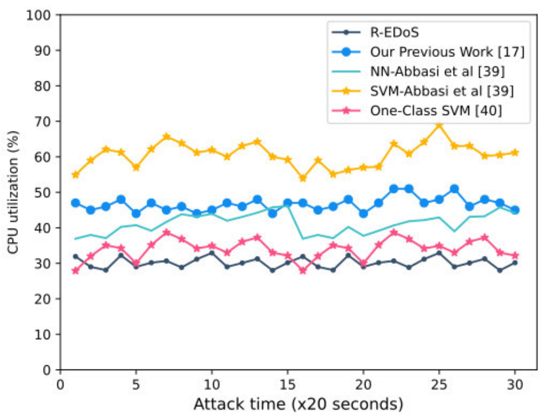

FIGURE 13. Compute usage according to attack rate.

into account by both *FPs* and *FNs*. As introduced in Table 8, our proposed scheme demonstrates its outstanding prediction ability with a 95.51% F1-score, which is higher than the second highest F1-score by 4.63%. The approaches Following NN-Abbasi *et al.*, SVM-Abbasi *et al.*, and one-class SVM achieve 76.58%, 85.4%, and 82.2%, respectively.

Regarding comparison among five attack scenarios (as shown in Figure 10 and Table 8), R-EDoS outperforms the others, and it can perform well on all five attack scenarios overall, even though it has some difficulties in detecting both YoYo and Slowloris attacks, due to their stealthy characteristic of mimicking normal traffic. Furthermore, the box plots of R-EDoS (in Figure 10) demonstrate the high stability of our proposed scheme's detection capability.

Through the comprehensive analyses above, it is clear that our multivariate time-series approaches (R-EDoS and our previous work [17]) outperform both deep-learning-based and machine-learning-based approaches (neural network, SVM [39] and one-class SVM [40]) in detecting EDoS attacks. Our proposal (R-EDoS) shows impressive results in detecting various types of attacks with low false-alarm rates.

C. ATTACK MITIGATION PERFORMANCE

Agrawal and Tapaswi [47] recently proposed some important measurements to assess the effectiveness of a defense scheme, such as accuracy, attack detection time, service response time, etc. Hence, the metrics given above are adopted in this paper. Note that all the metrics presented in this subsection and the following subsection were measured by us during a SYNFlooding attack with a fixed attack rate of 7000 req/s excluding the response time metric.

First, to measure how quickly a new attack is exposed, we created a function to calculate the average time for a new attack to be exposed during the simulated SYNFlooding attack we launched (as shown in Figure 11). As mentioned before, one-class SVM's main advantages are less time and storage space to run; it only requires 0.8 seconds on average to raise an alarm if it detects an EDoS attack, and definitely outperforms others in terms of detection time. However, the detection time difference between one-class SVM and our two approaches is not very significant because the data sequence length of R-EDoS and our previous work was only set at 100 and 250, respectively. That means the detection time of R-EDoS is only 0.91 seconds (and 0.97 seconds for our previous work).

Second, service response time is one of the most important metrics in evaluating any defense system [47]. This is defined as follows. Once a cloud user makes a request, then one corresponding response is sent back to the user. Figure 12 shows that the higher the attack rate, the longer the response time. Overall, the response time of all five solutions is short, varying between 11.1 ms and 32.6 ms. Specifically, R-EDOS, our previous work, and SVM-Abbasi *et al.* are similar in response time, gradually increasing in the 20 - 30 ms response time range.

D. RESOURCE CONSUMPTION AND COST CALCULATION

To demonstrate the lightweight mechanism of R-EDoS, resource consumption, including CPU and memory, is also provided along with the total estimated cost.

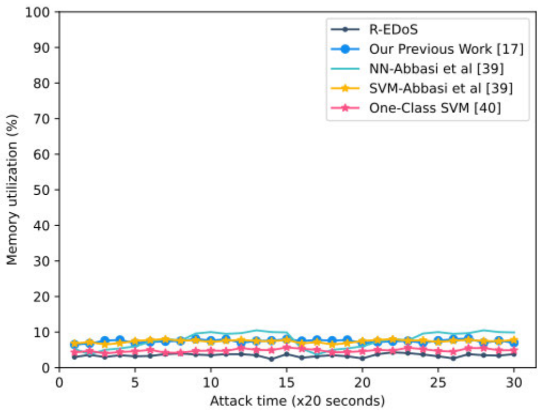

FIGURE 14. **Memory utilization in relation to attack rate.**

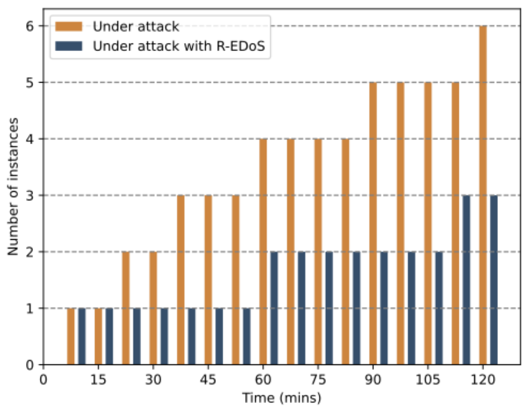

FIGURE 15. **Instances allocation with R-EDoS.**

As mentioned in section V.D, we placed all modules in an SDN controller node. CPU and memory usage were measured over time. For CPU usage, owing to setting the data sequence length = 100, R-EDoS consumes the lowest amount of CPU, varying between 28% and 33% over time, compared to other solutions (as presented in Figure 13), whereas, other solutions are highly resource-intensive and they tend to perform unreliably in that their CPU utilization widely fluctuates (i.e., SVM-Abbasi *et al.* varies between 55% and 69%). This wide fluctuation behavior may degrade the whole cloud network. As for memory usage, all solutions consume a low amount of the memory resource (under 9.4%), with R-EDoS only fluctuating between 2.86% and 3.7% (as presented in Figure 14). Evidently, R-EDoS outperforms other solutions, including our previous work, in terms of CPU and memory usage. This proves a great improvement of R-EDoS in resource usage compared to our existing work, as stated in the abstract.

Figure 15 shows the number of cloud instances allocated during the simulation of the SYNFlooding attack, with a fixed attack rate of 7000 req/s and the CPU upper threshold = 80%. The results show that when under attack, R-EDoS can save from 33% to 60% on the number of running instances. For example, 6 instances were launched after 120 minutes under the attack. By applying R-EDoS to the simulation, only 3 instances were spawned in the same amount of time. Or in the 105th minute, the number of instances under the attack with R-EDoS was only 2, compared to 5 instances without integrating R-EDoS during the attack.

Recently, Al-Haidari *et al.* [43] proposed a formula that can easily be used to estimate the future cost that a cloud user would have to pay for their resource usage based on this formula:

$$COST = (Price_{bw} * \lambda_{GB/s} + Price_{com} * S) * T \qquad (12)$$

where $Price_{bw}$ indicates the price of bandwidth for every 1 GB, $\lambda_{GB/s}(= \lambda_{legitimate} + \lambda_{malicious})$ is the arrival rate in which $\lambda_{legitimate}$ and $\lambda_{malicious}$ are the legitimate and attack traffic rates respectively, in requests per second, $Price_{com}$ implies the basic fee charged per hour for each instance, and T means the total number of hours that the cloud user intends to use cloud resources for their application, and S indicates the number of running instances.

Assume a cloud user needs to create a Web server for its business website. Therefore, the user chooses AWS [48] to deploy its application with system configurations as follows. Ubuntu Server version 20.04 runs on CPU t3.xlarge size (4 vCPUs, 2.5 GHz, 16 GiB memory, 8 GiB EBS with IOPS = 100 / 3000), the data transfer rate supports up to 5 Gigabit, and the on-demand price per hour in the Asia Pacific (Seoul) region is \$0.1344 [48]. Based on Equation 12, if the cloud user intends to run the business website for one month under EDoS attacks, doing so will cost an estimated \$654.372, where $Price_{bw} = 0.09$ [48], $Price_{com} = 0.1344$, $S = 6$, $\lambda_{GB/s} = 1$, and $T = 730$. Using our proposed scheme, the cloud user need only pay AWS with the amount of \$327.186 under EDoS attacks with R-EDoS integrated into the server, where $S = 3$ and $\lambda_{GB/s} = 0.5$. We estimate that in this scenario, the cloud user can avoid paying an extra \$327.186 each month for the EDoS attacks.

E. INTERPRETABILITY AND DEPLOYMENT DISCUSSION

First, during the simulation of different EDoS attacks, we interpret the experimental results using a feature importance technique [38] for each type of attack. The interpretability results are as follows.

- For the SYNFlooding attack, three key features that have the most contribution to the prediction are *no_syn_pkt*, *no_pkt_per_flow*, and *no_ack_pkt*.
- For the APIRequests attack, top-three features: *network.incoming.packets*, *memory_consumption*, and *disk_usage*.
- For the HTTPAttack attack, *src_ip*, *network_protocol*, *no_established_connections*, and *dst_ip* play the main role in detecting this type of attack.
- For the YoYo attack, the four main features that have the most contribution to the prediction are *no_requests_to_webserver*, *no_running_instances*, *src_ip*, *network_protocol*.

- For the Slowloris attack, *no_established_connections*, *no_ack_pkt*, and *network.incoming.packets* are two factors that greatly contribute to detect the Slowloris attack.

Second, during tuning of R-EDoS hyperparameters, we observed the sensitivity of the results to variations in the value of the hyperparameters. We found that the model becomes highly sensitive if we set **z**-space dimension at a high value. In this case, we found that the reconstruction probability was unable to find a good posterior, whereas the underfitting problem remains unchanged.

Third, in the case of a legitimate cloud user suddenly sends a large number of requests, which increase the values of some key features such as *network.incoming.packets*, *network.incoming.bytes*, *no_established_connections*, etc. R-EDoS may produce false-positive rates in this case. However, the dynamic threshold will dynamically adjust its value to reduce the false-positive rates because the constructed usual pattern of R-EDoS will change its pattern according to the values of the key features.

F. GENERAL DISCUSSION

Based on our comprehensive result analyses given above, we summarize some outstanding points that demonstrate the effectiveness of R-EDoS in detecting real EDoS attacks conducted in our practical testbed:

- R-EDoS not only produces low error rates but also yields high detection rates, accuracies, and F1-scores, which clearly show that it outperforms other solutions.
- Our proposed scheme can defense against some common EDoS attacks such as HTTPAttack, APIRequests, and SYNFlooding effectively, though it faces some slight difficulties in detecting both YoYo and Slowloris attacks.
- The proposed scheme needs only a very short time to detect a new attack.
- R-EDoS consumes a very low amount of CPU and memory.
- Using this defense scheme can save up 30% to 50% of the total cost that a cloud user is forced to pay for the unexpected cost coming from EDoS attacks.

VII. CONCLUSION AND FUTURE WORK

In this work, we propose a novel approach to deal with stealthy EDoS attacks known as a multivariate time-series anomaly detection system. This approach can apply to different network systems and adapts well. Second, we propose an intelligent, robust scheme called R-EDoS, which can process and extract valuable information from high-dimensional time-series data. This lays the foundations for an intelligent model to detect anomalous network traffic behaviors occurring in EDoS attacks easily. Our work not only can help cloud users to avoid paying for various EDoS attacks such as slow HTTP/SYN flooding, database API requests, Yo-Yo, and Slowloris, but also can allow cloud service provider to improve their services. In future research, we not only plan to enhance the proposed scheme but also plan to draw a detailed comparison with other EDoS defense systems, using more evaluation criteria for particular EDoS attacks.

REFERENCES

[1] J. Rubio-Loyola, A. Galis, A. Astorga, J. Serrat, L. Lefevre, A. Fischer, A. Paler, and H. Meer, "Scalable service deployment on software-defined networks," *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 84–93, Dec. 2011.

[2] Y. Li and M. Chen, "Software-defined network function virtualization A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[3] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 2, pp. 426–437, Jun. 2011, doi: 10.1109/TIFS.2011.2107320.

[4] W. Zhijun, L. Wenjing, L. Liang, and Y. Meng, "Low-rate DoS attacks, detection, defense, and challenges: A survey," *IEEE Access*, vol. 8, pp. 43920–43943, 2020, doi: 10.1109/ACCESS.2020.2976609.

[5] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602–622, 1st Quart., 2016.

[6] F. Z. Chowdhury, L. B. M. Kiah, M. A. M. Ahsan, and M. Y. I. B. Idris, "Economic denial of sustainability (EDoS) mitigation approaches in cloud: Analysis and open challenges," in *Proc. Int. Conf. Electr. Eng. Comput. Sci. (ICECOS)*, Palembang, Indonesia, Aug. 2017, pp. 206–211.

[7] T. G. Nguyen, T. V. Phan, B. T. Nguyen, C. So-In, Z. A. Baig, and S. Sanguanpong, "SeArch: A collaborative and intelligent NIDS architecture for SDN-based cloud IoT networks," *IEEE Access*, vol. 7, pp. 107678–107694, 2019, doi: 10.1109/ACCESS.2019.2932438.

[8] M. H. Sqalli, F. Al-Haidari, and K. Salah, "EDoS-shield—A two-steps mitigation technique against EDoS attacks in cloud computing," in *Proc. 4th IEEE Int. Conf. Utility Cloud Comput.*, Victoria, NSW, Australia, Dec. 2011, pp. 49–56.

[9] S. H. Khor and A. Nakao, "Spow: On-demand cloud-based eddos mitigation mechanism," in *Proc. 11th Int. Symp. Appl. Internet (SAINT)*. Piscataway, NJ, USA: IEEE Press, 2011, pp. 160–171.

[10] M. Masood, Z. Anwar, S. A. Raza, and M. A. Hur, "EDoS armor: A cost effective economic denial of sustainability attack mitigation framework for e-commerce applications in cloud environments," in *Proc. INMIC*, Lahore, Pakistan, Dec. 2013, pp. 37–42.

[11] F. Z. Chowdhury, M. Y. I. Idris, L. M. Kiah, and M. A. M. Ahsan, "EDoS eye: A game theoretic approach to mitigate economic denial of sustainability attack in cloud computing," in *Proc. IEEE 8th Control Syst. Graduate Res. Colloq. (ICSGRC)*, Shah Alam, Malaysia, Aug. 2017, pp. 164–169.

[12] P. Singh, S. Ul Rehman, and S. Manickam, "Comparative analysis of state-of-the-art EDoS mitigation techniques in cloud computing environment," 2019, *arXiv:1905.13447*. [Online]. Available: http://arxiv.org/abs/1905.13447

[13] A. Shawahna, M. Abu-Amara, A. Mahmoud, and Y. E. Osais, "EDoS-ADS: An enhanced mitigation technique against economic denial of sustainability (EDoS) attacks," *IEEE Trans. Cloud Comput.*, vol. 8, no. 3, pp. 790–804, Sep. 2020.

[14] N. Agrawal and S. Tapaswi, "A proactive defense method for the stealthy EDoS attacks in a cloud environment," *Int. J. Netw. Manage.*, vol. 30, no. 2, p. e2094, Mar. 2020, doi: 10.1002/nem.2094.

[15] M. Monge, J. Vidal, and L. Villalba, "Entropy-based economic denial of sustainability detection," *Entropy*, vol. 19, no. 12, p. 649, Nov. 2017.

[16] S. Weisber, "Yeo–Johnson power transformations," Dept. Appl. Statist., Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., Oct. 2001, pp. 1–4. [Online]. Available: http://www.stat.umn.edu/arc/yjpower.pdf

[17] P. T. Dinh and M. Park, "Dynamic economic-denial-of-sustainability (EDoS) detection in SDN-based cloud," in *Proc. 5th Int. Conf. Fog Mobile Edge Comput. (FMEC)*, Paris, France, Apr. 2020, pp. 62–69.

[18] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. 23rd Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, Bruges, Belgium, 2015, pp. 89–94.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: http://arxiv.org/abs/1412.3555

[20] D. P Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*. [Online]. Available: http://arxiv.org/abs/1312.6114

[21] J. Sun, X. Wang, N. Xiong, and J. Shao, "Learning sparse representation with variational auto-encoder for anomaly detection," *IEEE Access*, vol. 6, pp. 33353–33361, 2018, doi: 10.1109/ACCESS.2018.2848210.

[22] D. Jimenez Rezende and S. Mohamed, "Variational inference with normalizing flows," 2015, *arXiv:1505.05770*. [Online]. Available: http://arxiv.org/abs/1505.05770

[23] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in Web applications," 2018, *arXiv:1802.03903*. [Online]. Available: http://arxiv.org/abs/1802.03903

[24] S. Bhingarkar and D. Shah, "FLNL: Fuzzy entropy and lion neural learner for EDoS attack mitigation in cloud computing," *Int. J. Model., Simul., Sci. Comput.*, vol. 9, no. 6, Dec. 2018, Art. no. 1850049.

[25] P. T. Dinh and M. Park, "ECSD: Enhanced compromised switch detection in an SDN-based cloud through multivariate time-series analysis," *IEEE Access*, vol. 8, pp. 119346–119360, 2020, doi: 10.1109/ACCESS.2020.3004258.

[26] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2828–2837.

[27] G. Kitagawa and W. Gersch, "Linear Gaussian state space modeling," in *Smoothness Priors Analysis of Time Series* (Lecture Notes in Statistics), vol. 116. New York, NY, USA: Springer, 1996, pp. 55–65.

[28] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019, doi: 10.1109/TNNLS.2018.2846646.

[29] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *Proc. 31st Int. Conf. Very Large Data Bases*, 2005, pp. 697–708.

[30] C. J. Chu, "Time series segmentation: A sliding window approach," *Inf. Sci.*, vol. 85, nos. 1-3, pp. 147–173, 1995, doi: 10.1016/0020-0255(95)00021-G.

[31] L. de Haan and A. Ferreira, *Extreme Value Theory: An Introduction* (Springer Series in Operations Research and Financial Engineering). New York, NY, USA: Springer, 2006, pp. 65–126.

[32] M. R. Leadbetter, "On a basis for 'peaks over threshold' modeling," *Statist. Probab. Lett.*, vol. 12, no. 4, pp. 357–362, Oct. 1991.

[33] A. N. Iman and T. Ahmad, "Improving intrusion detection system by estimating parameters of random forest in Boruta," in *Proc. Int. Conf. Smart Technol. Appl. (ICoSTA)*, Surabaya, Indonesia, Feb. 2020, pp. 1–6, doi: 10.1109/ICoSTA48221.2020.1570609975.

[34] M. B. Kursa and W. R. Rudnicki, "Feature selection with the Boruta package," *J. Stat. Softw.*, vol. 36, no. 11, pp. 1–13, 2010.

[35] M. Cassel and F. L. Kastensmidt, "Evaluating one-hot encoding finite state machines for SEU reliability in SRAM-based FPGAs," in *Proc. 12th IEEE Int. Line Test. Symp. (IOLTS)*, Lake Como, Italy, Jul. 2006, p. 6, doi: 10.1109/IOLTS.2006.32.

[36] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Anal. Chem.*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964.

[37] J. Bi, H. Yuan, and M. Zhou, "Temporal prediction of multiapplication consolidated workloads in distributed clouds," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 4, pp. 1763–1773, Oct. 2019, doi: 10.1109/TASE.2019.2895801.

[38] A. Altmann, L. Tolosi, O. Sander, and T. Lengauer, "Permutation importance: A corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, May 2010.

[39] H. Abbasi, N. Ezzati-Jivan, M. Bellaiche, C. Talhi, and M. R. Dagenais, "Machine learning-based EDoS attack detection technique using execution trace analysis," *J. Hardw. Syst. Secur.*, vol. 3, no. 2, pp. 164–176, Jun. 2019.

[40] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 582–588.

[41] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proc. ACM SIGKDD Workshop Outlier Detection Description*. New York, NY, USA: Association Computing Machinery, 2013, pp. 8–15, doi: 10.1145/2500853.2500857.

[42] A. Bremler-Barr, E. Brosh, and M. Sides, "DDoS attack on cloud auto-scaling mechanisms," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, Atlanta, GA, USA, May 2017, pp. 1–9, doi: 10.1109/INFOCOM.2017.8057010.

[43] F. Al-Haidari, M. Sqalli, and K. Salah, "Evaluation of the impact of EDoS attacks against cloud computing services," *Arabian J. Sci. Eng.*, vol. 40, no. 3, pp. 773–785, Mar. 2015, doi: 10.1007/s13369-014-1548-y.

[44] P. Singh and S. Manickam, "Design and deployment of OpenStack-SDN based test-bed for EDoS," in *Proc. 4th Int. Conf. Rel., Infocom Technol. Optim. (ICRITO) (Trends Future Directions)*, Noida, India, Sep. 2015, pp. 1–5, doi: 10.1109/ICRITO.2015.7359327.

[45] N. Goix, "How to evaluate the quality of unsupervised anomaly detection algorithms?" 2016, *arXiv:1607.01152*. [Online]. Available: http://arxiv.org/abs/1607.01152

[46] E. Cambiaso, G. Papaleo, G. Chiola, and M. Aiello, "Slow DoS attacks: Definition and categorisation," *Int. J. Trust Manage. Comp. Commun.*, vol. 1, nos. 3–4, pp. 300–319, Jan. 2013.

[47] N. Agrawal and S. Tapaswi, "Defense mechanisms against DDoS attacks in a cloud computing environment: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3769–3795, 4th Quart., 2019, doi: 10.1109/COMST.2019.2934468.

[48] J. Polzehl and V. Spokoiny, "Propagation-separation approach for local likelihood estimation," *Probab. Theory Rel. Fields*, vol. 135, no. 3, pp. 335–362, Jul. 2006.

[49] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2014. [Online]. Available: https://onosproject.org

PHUC TRINH DINH (Member, IEEE) received the B.S. degree in information technology from Telecommunications University, Vietnam, in 2018, and the master's degree in information and communication from Soongsil University, Seoul, South Korea, in February 2021. His research interests include deep learning, reinforcement learning, big data, cloud computing, software-defined networks, and network security.

MINHO PARK (Member, IEEE) received the B.S. and M.S. degrees in electronics engineering from Korea University, in 2000 and 2002, respectively, and the Ph.D. degree from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, South Korea, in 2010. He is currently an Associate Professor with the School of Electronic Engineering, Soongsil University, Seoul. His current research interests include wireless networks, vehicular communication networks, network security, and cloud computing.