System Statistics Learning-Based IoT Security: Feasibility and Suitability

Fangyu Li, Aditya Shinde, Yang Shi, *Student Member, IEEE*, Jin Ye, *Senior Member, IEEE*, Xiang-Yang Li, *Fellow, IEEE*, and Wenzhan Song, *Senior Member, IEEE*

***Abstract*—Cyber attacks and malfunctions challenge the wide applications of Internet of Things (IoT). Since they are generally designed as embedded systems, typical auto-sustainable IoT devices usually have a limited capacity and a low processing power. Because of the limited computation resources, it is difficult to apply the traditional techniques designed for personal computers or super computers, like traffic analyzers and antivirus software. In this paper, we propose to leverage statistical learning methods to characterize the device behavior and flag deviations as anomalies. Because the system statistics, such as CPU usage cycles, disk usage, etc., can be obtained by IoT application program interfaces, the proposed framework is platform and deviceindependent. Considering IoT applications, we train multiple machine learning models to evaluate their feasibility and suitability. For the target auto-sustainable IoT devices, which operate well-planned processes, the normal system performances can be modeled accurately. Based on time series analysis methods, such as local outlier factor, cumulative sum, and the proposed adaptive online thresholding, the anomalous behaviors can be effectively detected. Comparing their performances on detecting anomalies as well as the computation sources required, we conclude that relatively simple machine learning models are more suitable for IoT security, and a data-driven anomaly detection method is preferred.**

***Index Terms*—Anomaly detection, deep learning, failure and intrusion detection, Internet of Things (IoT), machine learning.**

I. Introduction

LIMITED computation ability and small capacity of Internet of Things (IoT) devices result in poor built-in security and forensics capabilities [1]. Increasing connectivity between various devices through the Internet, IoT leverage heterogeneous edge nodes (a diverse set of computing resources), which causes the vulnerability to attacks. Note that, in this paper, we focus on the auto-sustainable, i.e., self-organized [2], IoT devices with limited computation and communication resources. Unlike traditional computer systems, it is difficult to frequently update and patch these systems and any such operation requires months of advanced planning [3], [4]. Thus, traditional signature-based intrusion detection systems are difficult to implement on these embedded devices.

Statistical learning-based anomaly detection has been widely applied and studied in the computer security domain [5]–[7]. Assuming normal behavior of a device follows a pattern, thus, models can be trained to learn and predict if the observed behavior is normal or abnormal. For example, Al-Garadi *et al.* [8] made a comprehensive survey on learning-based methods developed for IoT security. In real applications, some IoT devices are designed to operate self-optimizations or autonomous system adaptations when unnoticed for certain time length [9]. If they are known beforehand, these processes can be considered during the model training. Or, system administrators can update the model when these system behaviors are recognized as anomalies. Anomaly detection is well suited for IoT security since it does not impose any heavy requirements such as installation of special devices in the network [10]. Nowadays, the system statistics can be collected by the application program interfaces (APIs) associated with the IoT devices, so the system statistics-based anomaly detection has broad applications. In case of more powerful devices like Beagle Bone Black (BBB), Raspberry Pi's or other such embedded devices which have an operating system on board, we can even run the model locally. Since different kinds of anomaly detection models can be trained according to the applications and hardware configurations, learning-based security methods have a better generalized property in comparison to traditional signature-based intrusion detection systems [8].

Various architectures have been proposed for IoT systems [11]. Khan *et al.* [12] described a general architecture and divided IoT into five layers: 1) perception; 2) network; 3) middleware; 4) application; and 5) business. Apart from known attack vectors, IoT devices in the perception layer are also vulnerable to various physical attacks since they usually operate in external environments [13], [14]. An attacker may steal or tamper with the data that sensors collect. On the network layer, the system is vulnerable to man-in-the-middle attacks, denial of service (DoS) attacks [15], etc. The application layer is in many cases vulnerable to the same attacks that plague traditional computer systems, i.e., attacks ranging from structured query language injections to authentication failures depending on the application. The same is true in the case of network layer devices which use the transmission

Manuscript received August 14, 2018; revised November 17, 2018, January 8, 2019, and January 23, 2019; accepted January 29, 2019. Date of publication February 4, 2019; date of current version July 31, 2019. The research was partially supported by NSF-1663709 and Southern Company. *(Corresponding author: Fangyu Li.)*

F. Li, A. Shinde, Y. Shi, J. Ye, and W. Song are with the Center for Cyber-Physical Systems, University of Georgia, Athens, GA 30602 USA (e-mail: fangyu.li@uga.edu; adityas@uga.edu; yang.atrue@uga.edu; jin.ye@uga.edu; wsong@uga.edu).

X.-Y. Li is with the School of Computer Science, University of Science and Technology of China, Hefei 230026, China (e-mail: xiangyangli@ustc.edu.cn).

Digital Object Identifier 10.1109/JIOT.2019.2897063

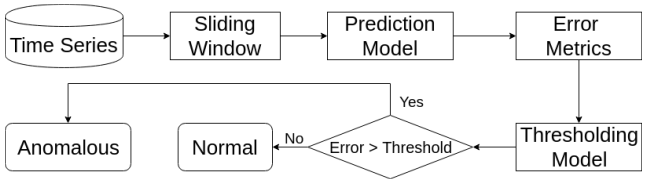

Fig. 1. General system operation workflow for IoT security.

control protocol/Internet protocol (TCP/IP) protocol stack for communication, which is already a very broad attack surface. In this paper, we focus on the IoT security issues on fog services, which are based on small and low-power computers, such as Raspberry Pis, to be clustered together as independent and portable mini-clouds and deployed in indoor or outdoor environments [16].

In this paper, we introduce a statistical learning-based anomaly detection technique to monitor the operation of IoT devices and detect possible cyber attacks or malicious activities. Our technique uses simple system statistics like CPU usage, memory consumption, network throughput, etc. to model the normal behavior. Collecting these data can be relatively trivial compared to running security software locally. Furthermore, because system statistics can be easily collected by APIs designed for different IoT platforms, our proposed approach can be viewed as a platform/device independent solution. Our contributions are as follows.

1) We use different machine learning models to predict the behavior of IoT devices and then observe the deviations from the expected behavior using time series analysis methods. A general workflow is shown in Fig. 1.
2) Our method can be used for general failure detection since failures have not been modeled by the prediction model, so the anomalous behaviors can be detected.
3) We propose a time series analysis-based anomaly detection method. The proposed method achieves better performances than the traditional methods.
4) Comparing the performances and required computation resources, we discuss the suitabilities of different learning models with IoT security, and simpler models with acceptable performances are adapt to IoT applications.

II. Related Work on IoT Security

In this section, the security threats in IoT and possible solutions are introduced. Our IoT security solutions are related to system usage analysis and the existing cyber-physical systems (CPSs) security solutions. Most detection techniques identifying these attacks are specific to the attack type and try to adapt traditional security solutions like intrusion detection systems to IoT. Here, we introduce the previous work in the related domain to clarify our motivations and innovations.

A. Cyber Security in IoT

Granjal *et al.* described protocols at various layers of the IoT stack which are used to ensure reliable and authorized communication between devices in [17]. This is analogous to the implementation of various protocols at every layer of the TCP/IP stack. However, secure communication protocols alone are not enough to protect networked devices. da Silva *et al.* proposed a rule-based intrusion detection system for wireless sensor networks in [18]. The challenge in adapting this system for general IoT is to design rules in order to cover numerous attacks, which may not generalize well. Raza *et al.* [19] proposed a light weight intrusion detection system specially designed for IoT. And, they also proposed a light weight distributed firewall for IoT networks. Their work, however, is limited to detecting specific types of attacks on the network layer, leaving a large number of attacks out of scope of detection.

Another important threat type in IoT networks is DoS attacks. Kasinathan *et al.* [15] summarized the following types of DoS attacks that IoT devices are vulnerable.

1) Jamming: Jamming attacks usually take place at the lower layers of the IoT architecture. The attacker sends forged packets to create collisions and force the device to drop legitimate packets [15]. Raymond *et al.* [20] discussed the effects of various DoS attacks on wireless sensor networks. Xu *et al.* [21] analyzed the efficiency of detecting these attacks using parameters like packet delivery ratio, signal strength, and carrier sensing time. They found that detection of a jammer is difficult using these measurements individually.

2) Routing Attacks: Routing attacks employ spoofing techniques to alter routing information in order to change the network structure. Homing attack, desynchronization, hello flooding, and black hole attacks are a few routing attacks against IoT devices [22]. In hello flooding, the attacker replays hello messages using a high power transmitter. Even nodes which are not in range of the broadcasting node will try and connect to it. This causes unreliable forwarding of traffic. In the black hole attack, the attacker makes himself a part of the network, then changes routing information to make most traffic passes through the attacker's node and drops the packets [22].

B. Resource Usage Analysis and Anomaly Detection

Resource usage patterns have been modeled for security and behavior analysis [23]. Caviglione *et al.* [24] discovered malicious processes communicating through covert channels, where neural networks (NNs) and decision trees were used to detect anomalies in usage patterns. However, their work mainly targeted on smart phones, which run a variety of applications unlike IoT devices that are geared toward specific type of applications. Sorkunlu *et al.* [23] used resource usage patterns to detect anomalies in high performance computing (HPC) clusters. Their technique is geared toward computer clusters but not IoT devices. In case of IoT devices, the resource usage patterns are likely periodic since they run a specific application. And, all features will vary periodically. The values will oscillate around the mean. This creates a bimodal distribution of data points. So it is difficult to find a low dimensional approximation since the principal components will always align toward these periodic time series due to their high variance.

Anomaly detection in CPS and sensor networks is primarily applied with the objective of detecting faulty measurements

and failures. Janakiram *et al.* [6] used Bayesian belief networks to detect faulty measurements, which has to be designed manually. Also, faulty measurement may not necessarily detect faults or security breaches. A sensor may still transmit correct data while being compromised. Idé *et al.* [7] used correlation between multiple sensors to form neighborhood graphs. However, their method treated the system as a collection of sensors. They did not evaluate the deviation of individual sensors. Also, since they measured the difference between neighbors, the anomaly may go undetected if it is correlated across multiple nodes. Idé and Kashima [25] proposed an eigen decomposition-based anomaly detection method mainly to servers and computer devices. Their technique could be adopted to detect anomalies on IoT networks. But it required the construction of dependency graphs and modeling the activity in terms of activity vectors. Also the dependency graphs have to be constructed from overall network activity which requires a single node analyzing the network traffic.

III. Algorithm and System Design

The purpose of system statistics prediction is to model the system processes which can produce the given time series. The modeling error can be calculated as the difference between the predicted and observed value. Once the model is sufficiently trained, a high error would signify a deviation from expected behavior. Thus, the error sequence at each time step can be thresholded using a time series analysis method and any point above that can be treated as an anomaly.

The prediction of the next step X^{pred} can be obtained based on the previous value X^{past}

$$X^{\text{pred}} = f\left(X^{\text{past}}\right) + \epsilon \tag{1}$$

where $f(\cdot)$ represents a trained model, ϵ is random noise. The prediction error E in the time series modeling is then calculated as the difference between the prediction and observation. The anomaly detection process based on thresholding $\tau(\cdot)$ can be broadly described as

$$\tau(\text{data}) = \begin{cases} \text{anomalous}, & \text{if } E > \text{threshold} \\ \text{normal}, & \text{if } E < \text{threshold}. \end{cases} \tag{2}$$

A. Statistical Learning Models

A predictive model gives a value of the time series variables at the next time sample according to the previous time steps. There are many ways to model a time series behavior based on the values of visual and hidden variables. In this paper, we adopt statistical learning methods to model the IoT system. From simplicity to complication, linear regression, NNs, and recurrent NNs (RNNs) models are introduced, whose performances are compared in Section V.

1) Linear Regression: The linear regression models assume linear relations between the target sample and the previous samples. The Markov property is a prerequisite, i.e., the current sample is only dependent on previous i samples. For instance, a second order auto regression model can be written as $x_i = w_0 + w_1 x_{i-1} + w_2 x_{i-2} + \epsilon$, where w_0 is a constant, w_1 and w_2 are the regression coefficients. Similarly, a 50th order auto regression model can be written as $x_i = w_0 + \sum_{n=1}^{50} w_i x_{i-n} + \epsilon$, which can be more appropriately represented in form of matrices

$$\mathbf{X}^{\text{pred}} = \mathbf{X}\mathbf{W} + \mathbf{C} \tag{3}$$

where $\mathbf{X}^{\text{pred}} \in \mathcal{R}^{t\times 1}$, $\mathbf{X} \in \mathcal{R}^{t\times n}$, and $\mathbf{W} \in \mathcal{R}^{n\times 1}$. The model is parameterized by the learned weights $\mathbf{W}$ can be represented as a linear function f in (1).

The objective of learning is to obtain accurate estimates of the regression coefficient parameters w_i assure that the error between the predicted value and true value is minimum. We use the mean absolute error (MAE) as a metric to measure how well our model fits the true data, whose definition is $\mathbf{MAE} = L(\mathbf{W}) = (1/n)|(\mathbf{X}^{\text{pred}} - \mathbf{X}^{\text{true}})|$, where $\mathbf{X}^{\text{true}}$ is the observed value. The algorithm uses an iterative method, and stops when the error converges. The weight update based on the gradient descent can be expressed as $\mathbf{W} \coloneqq \mathbf{W} - \alpha(\partial L/\partial \mathbf{W})$. Then, predictions are estimated by (3).

2) Neural Networks: As machine learning models, NNs are good at modeling nonlinear relationships. A general NN architecture is shown in Fig. 2. The NN essentially operates in the same way as (3), but models a nonlinear function. The nonlinearity is induced by using activation functions on the layers. For our experiments, we use the rectified linear unit activation function, which can be written as $r(x) = \max(0, x)$. In the output layer, we use a sigmoid activation function as $f(x) = e^x/(1+e^x)$. In our experiments, we try NN with different hyper parameters and evaluate their performances in the time series modeling (details are shown in Section V).

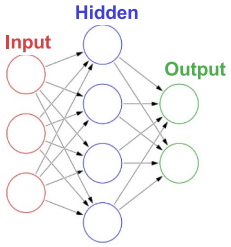

Fig. 2. Fully connected NN model with one single hidden layer [26].

3) Recurrent Neural Network: A RNN is similar to a NN but it has connections across successive time steps. Thus, it is better at modeling relationships in time series data. RNNs are essentially fully connected NNs unrolled over a specific number of time steps. Vanilla RNNs are difficult to train since they suffer from problems like vanishing gradient [27]. Hence, several improvements to the vanilla RNN cell have been proposed. For our experiments, we used RNN with a gated recurrent unit (GRU) cell, shown in Fig. 3, which can be summarized as

$$z_t = f_g(W_z x_t + U_z h_{t-1} + b_z) \tag{4}$$

$$r_t = f_g(W_r x_t + U_r h_{t-1} + b_r) \tag{5}$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ f_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \tag{6}$$

where x_t is the t time input, h_t, z_t, and r_t are the output vector, update gate vector, and reset gate vector, respectively. W, U, and b are parameter matrices and vector [28]. A single RNN

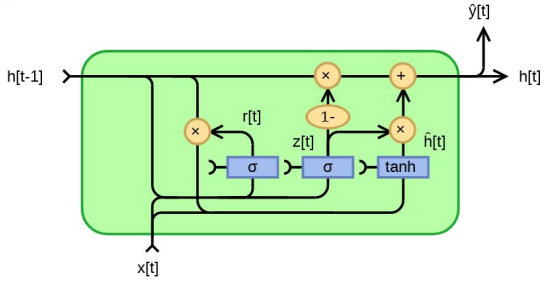

Fig. 3. GRU cell used in RNN [29].

layer is built by stacking GRU cells such that the h_t of one cell is the input of its adjacent cell. A full RNN is constructed by stacking these layers on top of each other.

In terms of the computational complexity, we compare the computation costs of linear regression, NN and RNN models. Calling n the number of observations and p the number of weights, the overall complexity of linear regression should be $\mathcal{O}(n^2p + p^3)$. For a NN layer with l_{i-1} input nodes, l_i output nodes, t training examples, and m epochs, the computational complexity is $\mathcal{O}(mt(l_{i-1} * l_i))$. For a RNN with H hidden units, I forward-connected units, K output units and m epochs, the computational complexity is $\mathcal{O}(m(IH + H^2 + HK))$. Note that when m is large, NN and RNN are much heavier than linear regression.

B. Statistical Anomaly Detection

Based on prediction models, we obtain the differences between observations and predictions. In order to identify the anomalous events, the prediction errors are analyzed by statistical methods. Thresholding-based anomaly detection is an important component of the system for deciding if a particular point is actually an anomaly or just noise. If the error probability distribution is known, thresholding can easily be done by checking if the number of outliers exceeds the percentage of the distribution. For instance, if the data are assumed to follow a normal distribution, sample points which are greater than $\mu+3\sigma$ or smaller than $\mu-3\sigma$ (μ is the mean value and σ is the standard deviation), are very likely to be anomalies (the probability is greater than 99.7%). Here, we introduce three different thresholding methods, and their performances will be later compared in Section V.

1) Local Outlier Factor: The local outlier factor (LOF) measures the local deviation of a given data point A with respect to its neighbors $N_k(A)$ based on the reachability distance [30]. The local reachability densities are then compared with those of the neighbors as $\text{LOF}_k(A) := [(\sum_{B \in N_k(A)} [\text{lrd}(B)/\text{lrd}(A)])/|N_k(A)|]$, where $\text{lrd}(\cdot)$ is the local reachability density. If LOF value is larger than 1, the sample point will be recognized as an anomaly.

2) Cumulative Statistics Thresholding: Cumulative sum control chart (CUSUM) is another time series change point detection method [31], which calculates a cumulative sum, defined as a process x_n with assigned likelihood weights ω_n. When the CUSUM value exceeds a certain threshold value, a change point is detected.

Algorithm 1 Proposed AOT Algorithm

Input: Prediction error
Output: Detected anomalies
Initialize:
Anomalies, expanding mean, threshold as EmptyArray, $\mu = 0$, $\sigma = 0$
for i = 1 to Error.length **do**
  if $i < 100$ and Error[i] > threshold **then**
    expanding mean[i] = μ
    threshold[i] = $\mu + (\lambda \times \sigma)$
    Anomalies[i] = 1
    Update λ with small forgetting coefficients
  else
    μ, σ = Compute running mean, standard deviation.
    Update λ with large forgetting coefficients
  end if
end for
return Anomalies

3) Adaptive Online Thresholding: As mentioned above, a classic threshold for normal distribution data is 3σ. However, the real applications do not always follow the normal distribution. Since 99.8% values in our observations are less than 5σ away from the mean, any value outside the range of $|\mu \pm 5\sigma|$ is statistically likely to be an anomaly.

Thus, an adaptive online thresholding (AOT) method is proposed to obtain a data-driven threshold based on online statistics. To compute μ and σ online, we use Welford's algorithm

$$\mu_n = \frac{(n-1)\mu_{n-1} + x_n}{n} = \mu_{n-1} + \frac{x_n - \mu_{n-1}}{n}$$
$$\sigma_n^2 = \frac{M_{2,n}}{n} = \frac{M_{2,n-1} + (x_n - \mu_{n-1})(x_n - \mu_n)}{n} \quad (7)$$

where $M_{2,n}$ is the second order moment.

In order to allow the historical data to make different significances, we propose to update thresholds using different forgetting factors according to the online μ and σ. The initial threshold is 5σ according to Chebyshev inequality. If a change point (anomaly) is detected, a small forgetting coefficient is adopted in the local statistics calculation to balance the influence from the anomaly. Otherwise, a large forgetting coefficient is used to characterize more temporal properties. The algorithm is written in detail in Algorithm 1.

Comparing LOF, CUSUM, and AOT, the computational complexity of the statistical anomaly detection step is much smaller than the statistical learning step discussed in the last section. Specifically, if the number of observations is n, the complexity of LOF is $\mathcal{O}(n^2)$, while CUSUM and AOT are both $\mathcal{O}(n)$. Thus, the computation costs of the learning model training should be given more attentions when considering the computation overhead.

IV. Experiment

A. System Setup

In order to simulate a real IoT system operation, we run a program, which samples random data with a fixed interval and carries a variety of signal processing, storage, compression, and transmission operations. The system consists of 12 BBB wireless boards connected to a router in a LAN topology. Besides, the network contains one aggregation server which collects and stores the statistics sent by BBB nodes.

B. Attack Design

Recently, a comprehensive IoT attack survey has been made by [32]. There are various attacks leading to different system statistics performances. To evaluate our proposed method, we design a set of cyber attacks, which will be discussed in Section V-C, as follows.

1) *Unauthorized Access:* Since the target IoT devices are auto-sustainable, the log-in actions are well planned by the system administrator. So the accidental log-in usually lacks of authentication. Then sensitive data manipulation is the main goal of such attacks [33].
2) *Port Scan:* The attackers use a port scan tool to send a message to each port to test if the port has some weaknesses, which can be compromised [34]. The behavior of this attack can be reflected using the network properties.
3) *Virus:* Virus services attack system weaknesses in IoT devices. Those programs may send out sensitive data or run processes [35]. The system performances of virus services are related to the virus action types.
4) *Flood:* As a kind of DoS attacks, an attacker sends a large number of packages to make the device unreachable [36]. If the flood attack happens to the network communication, a large number of received packages can be observed.

C. Data Collection

We use "collectd," a statistics collection daemon for Linux/Unix systems, to collect various system statistics from the 12 BBB nodes [37]. All of these statistics are collected through collectd by configuring its plug-ins. Each of these individual statistics may contain multiple attributes. These are recorded and time stamped accordingly by collectd. The statistics are then sent to the aggregation server every 5 s. This is a reasonable sampling rate as it does not excessively load the system. We configure collectd to gather data on the statistics listed in Table I. The aggregation server has tools to store time series data efficiently and also to create visualizations for analytics. For storing the data, we use InfluxDB, a time series database [38], which makes it possible to query and group data for various devices efficiently. We use the HTTP API to query for batches of data in order to create a database for training the machine learning models.

D. Data Preprocessing

The collected data have different amplitude ranges. We normalize the time series data between 0 and 1. Scaling can be very sensitive to outliers since the maximum and minimum of any time series are used for deciding the scaling factor. In presence of outliers, for instance abnormally high readings, the other data points may get scaled to a very narrow range. In order to prevent this, we clip these series to lie between certain values and then scale them between 0 and 1, as shown in Fig. 4. Some time series in Table I are redundant, i.e., they are flat or constant, so not modeled in our experiments.

E. Model Training

We use linear regression, NN and RNN models to predict system behaviors. All the models are implemented using Keras [39], a deep learning framework with TensorFlow [40] on the back end. MAE is used as a metric to judge how well the model has fit the data. We use early stopping to decide if the model has converged, which keeps a track of the performance on the validation set. Once the error stops decreasing, the model training stops. We use fivefold cross-validation in the training procedure, which means random 80% of the data for training and the rest 20% for validation.

TABLE I
SYSTEM USAGE STATISTICS USED IN OUR EXPERIMENTS

Attribute	Time Series
CPU Usage	idle, interrupt, nice, softirq, steal, system, user, wait
Interface TX	lo if_dropped, lo if_errors, lo if_packets, lo if_octets, wlan0 if_dropped, wlan0 if_errors, wlan0 if_packets, wlan0 if_octets
Interface RX	lo if_dropped, lo if_errors, lo if_packets, lo if_octets, wlan0 if_dropped, wlan0 if_errors, wlan0 if_packets, wlan0 if_octets
Disk IO time	mmcblk1, mmcblk1p1, mmcblk1boot0, mmcblk1boot1

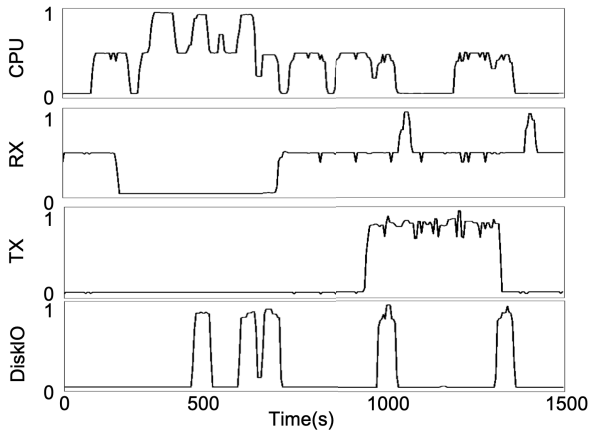

Fig. 4. Normalized system statistics after preprocessing.

V. RESULTS AND EVALUATION

A. Model Suitability

We train and test linear regression, NN, and RNN models with various window sizes, respectively. The MAE in modeling the time series for all models is summarized in Table II. The results show that models perform best when the window sizes are around 20 time samples. Fig. 5 depicts how the error changes as the window size varies. From the figure and table, it is clear that NNs perform the best. The RNNs with GRU units usually perform better at time series prediction. However, in our experiments, they show poor performances. This may be because the time series is too simple to learn for the RNNs or the data may not be sufficient to converge. Another observation is that the error rate generally stops improving when window size is larger than 50 time samples. Thus, it can be concluded that models can learn the device time series patterns based on the previous 50 time sample data.

While it is certainly desirable to have a model with very low error rate, performance can be traded off for lower complexity and computation requirements. The plots from Fig. 5 clearly

TABLE II
MAE WITH DIFFERENT MODELS AND TIME WINDOW SIZES

Model\Window size	MAE $\times(10^{-2})$						
	2	5	10	20	50	100	200
Linear	1.23	0.78	0.76	0.74	0.77	0.83	0.96
NN (1 layer)	0.80	0.69	0.69	0.70	0.68	0.76	0.76
NN (2 layers)	0.80	0.66	0.69	0.67	0.67	0.70	0.72
NN (3 layers)	0.83	0.70	0.70	0.71	0.71	0.75	0.77
RNN (1 layer)	0.88	0.77	0.79	0.80	0.82	1.09	0.87
RNN (2 layers)	0.88	0.77	0.78	0.79	0.78	0.95	1.11
RNN (3 layers)	0.89	0.77	0.78	0.77	0.75	0.74	0.77

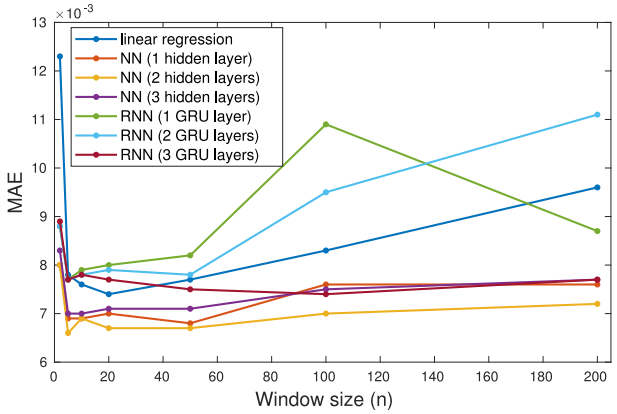

Fig. 5. Prediction error versus window size of different models.

show that NNs perform better. But the linear regression model also performs quite well. Additionally, the linear regression model is also much simpler than NN, so there is no problem to run linear regression model on IoT devices.

B. Anomaly Detection Method Comparison

In Section III-B, we introduce anomaly detection methods. Here, we use the F_1 score to quantitatively evaluate how well the methods detect anomalies with different durations. The F_1 score is defined as $F_1 = [(2 \times \text{precision} \times \text{recall})/(\text{precision} + \text{recall})]$, where the precision and recall scores are determined as precision $= [\text{tp}/(\text{tp} + \text{fp})]$, recall $= (\text{tp}/[\text{tp} + \text{fn}])$. And tp, fp, and fn stand for true positives, false positives, and false negatives, respectively.

Given a reasonably accurate detector, as the duration of the anomaly increases, the F1 measure of the anomaly should increase since the overlap between the true anomaly and prediction increases. In such a case, the model with the better F1 measure for a given time window is better for anomaly detection. Fig. 6 shows the F1 and recall scores of the model for various anomaly durations. The method using AOT performs better than CUSUM and LOF. Recall score measures the true positives. Note that the recall score is highest for LOF method. However, the F1 score is very poor for the same method. This shows that the LOF method has a lot of false positives. A probable reason for such poor performance of LOF is likely to be the violation of the normality assumption. The LOF method assumes normally distributed data. But the real distribution is skewed and has a heavy tail. In contrast, the AOT method decides the decision boundary based on the percentiles computed on the real distributions. Hence, it performs the best among other techniques. The actual F1 and recall

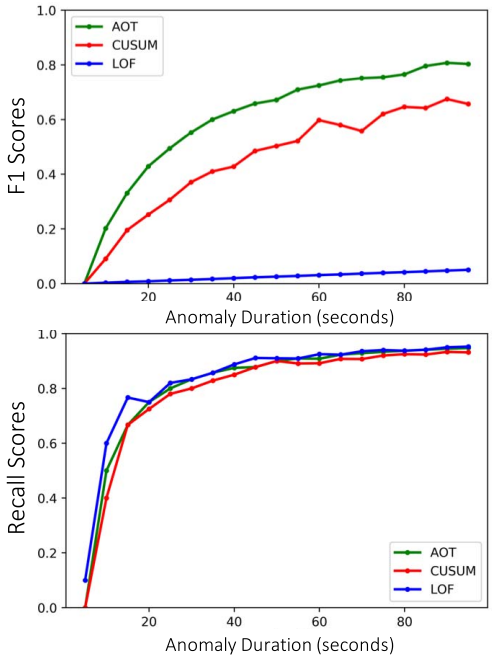

Fig. 6. F1 and recall scores from AOT, CUSUM, and LOF using a linear model with 20 time steps.

TABLE III
F1 AND RECALL SCORES FROM DIFFERENT ANOMALY DETECTION ALGORITHMS WITH ANOMALY DIFFERENT DURATIONS

Duration (seconds)	F1 scores			Recall scores		
	LOF	CUSUM	AOT	LOF	CUSUM	AOT
5	0.0005	0.0000	0.0000	0.1667	0.0000	0.0000
10	0.0030	0.0984	0.2000	0.5333	0.4500	0.5000
15	0.0058	0.1818	0.3287	0.6778	0.6111	0.6556
20	0.0088	0.2447	0.4208	0.7750	0.7167	0.7417
25	0.0115	0.3025	0.4979	0.8067	0.7733	0.8000
30	0.0148	0.3773	0.5566	0.8667	0.8111	0.8333
35	0.0171	0.4087	0.5950	0.8619	0.8476	0.8571
40	0.0199	0.4284	0.6204	0.8792	0.8542	0.8750
45	0.0227	0.4631	0.6602	0.8926	0.8593	0.8852
50	0.0254	0.4930	0.6801	0.9000	0.8867	0.9000
55	0.0283	0.5262	0.7026	0.9121	0.8970	0.9091
60	0.0313	0.5375	0.7127	0.9250	0.9056	0.9167
65	0.0340	0.5815	0.7461	0.9308	0.9103	0.9231
70	0.0369	0.6017	0.7500	0.9381	0.9190	0.9286
75	0.0393	0.6150	0.7595	0.9333	0.9178	0.9333
80	0.0425	0.6228	0.7847	0.9479	0.9271	0.9375
85	0.0448	0.6326	0.7837	0.9431	0.9333	0.9412
90	0.0476	0.6524	0.7876	0.9463	0.9315	0.9444
95	0.0507	0.6785	0.8030	0.9561	0.9386	0.9474

scores for all three techniques are shown in Table III. Note that, the recall of AOT equals 0.5 when the anomaly duration is 10 s, which means the anomaly detection has started working once we have more than 2 samples available (the interval is 5 s).

C. Anomaly Detection Case Study

1) Unauthorized Access: The first synthetic attack simulates user login and making changes. The prediction results are generated using a linear regression model with window size of 10 time samples. Fig. 7 shows an example of unauthorized access anomaly detection in system operations. Fig. 7(a) shows the prediction error with true anomaly marked. The Fig. 7(b) shows the anomaly detection results using LOF,

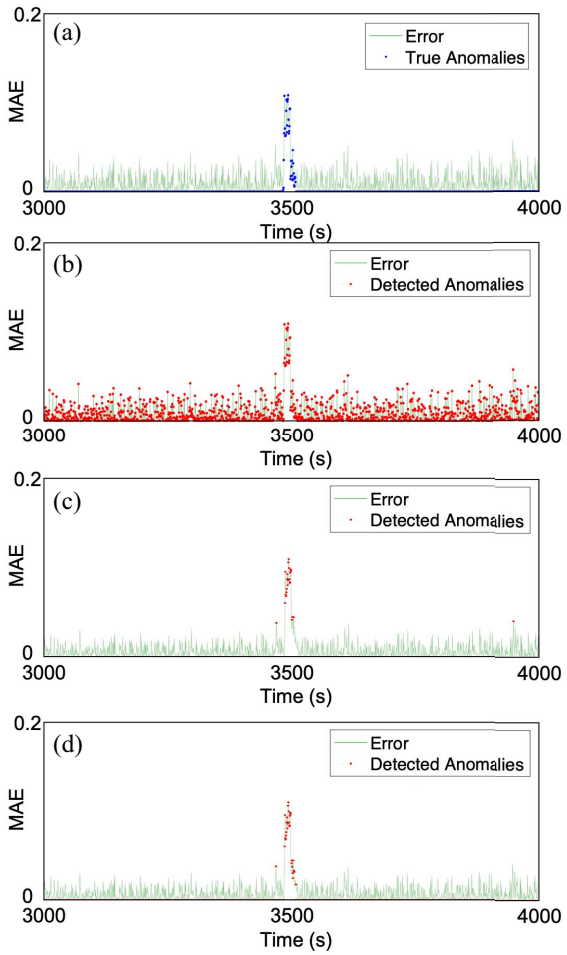

Fig. 7. Unauthorized access attack detection comparison among different thresholding methods. (a) Prediction error. Detection results from (b) LOF, (c) CUSUM, and (d) AOT.

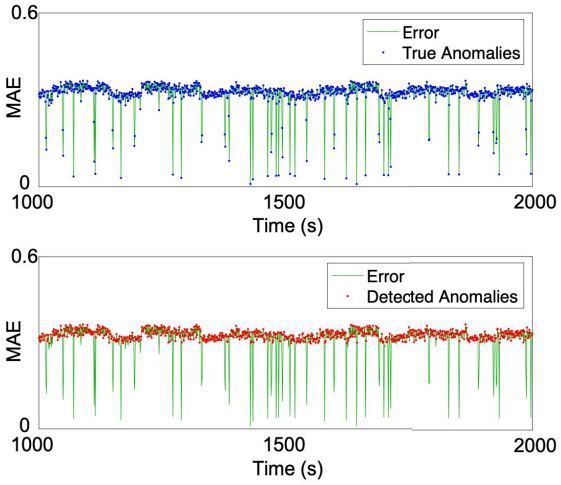

Fig. 8. Port scan attack detection example.

while Fig. 7(c) and (d) are those from CUSUM and AOT. It is clear that the proposed AOT method has the best performance, because it has the minimum false alarms and detects the most anomalies, which is exactly the reason why AOT has the highest F1 score. In addition, with further investigation, we find that this anomaly was caused due to the disk IO being extremely high as a result of excess usage.

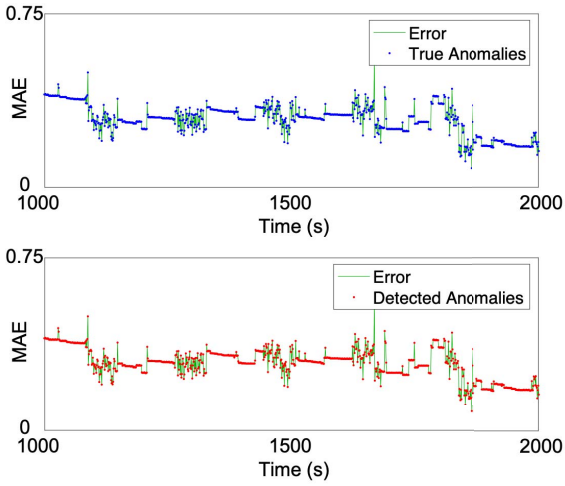

Fig. 9. Virus detection example.

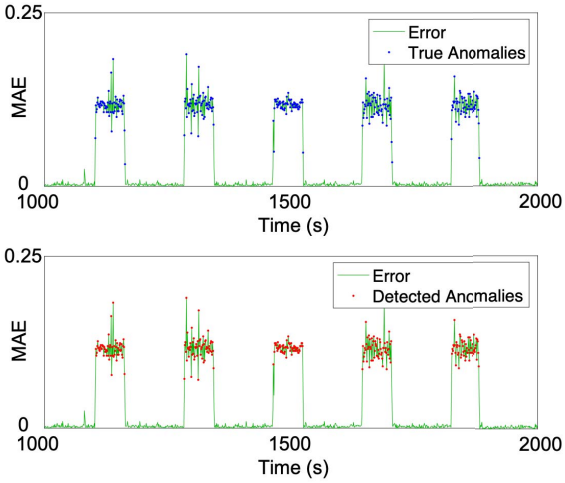

Fig. 10. Flood attack detection example.

2) Port Scan: A port scan attack is simulated using the Nmap tool [41], shown in Fig. 8. Since the simulated attack is continuous, the system behavior generates relatively large MAE, which is also constant to certain value according to the attack strength.

3) Virus: The simulated virus affects multiple system statistics. So Fig. 9 shows the prediction error behavior is quite complicated. However, since our proposed method models normal behaviors, all the abnormal performances can be detected.

4) Flood: We implement the network flood attacks with a fixed interval. The periodic anomalies are detected by the proposed method in Fig. 10.

VI. Conclusion

Cyber attacks on auto-sustainable IoT devices leave behavior traces in system statistics. This paper proposes a general anomaly detection-based framework for detecting cyber attacks and malicious behavior in IoT devices with limited

computation and communication resources. The main advantage of this technique is the capability to be implemented on any device from a small sensor to a laptop scale computation node. Our approach is especially suitable for auto-sustainable IoT since these devices usually run specific types of applications unlike general desktop computers. We have shown that statistical techniques and machine learning models can be used effectively to detect attacks on IoT devices. We have also found that it is possible to learn the behavior models of IoT devices using relatively simple machine learning models.

REFERENCES

[1] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.

[2] Ö. U. Akgül and B. Canberk, "Self-organized Things (SoT): An energy efficient next generation network management," *Comput. Commun.*, vol. 74, pp. 52–62, Jan. 2016.

[3] A. Cárdenas *et al.*, "Challenges for securing cyber physical systems," in *Proc. Workshop Future Directions Cyber Phys. Syst. Security*, vol. 5, 2009, pp. 1–7.

[4] C. Tankard, "The security issues of the Internet of Things," *Comput. Fraud Security*, vol. 2015, no. 9, pp. 11–14, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361372315300841

[5] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Comput. Netw.*, vol. 51, no. 12, pp. 3448–3470, 2007.

[6] D. Janakiram, V. A. Reddy, and A. V. U. P. Kumar, "Outlier detection in wireless sensor networks using Bayesian belief networks," in *Proc. 1st Int. Conf. IEEE Commun. Syst. Softw. Middleware Comsware*, 2006, pp. 1–6.

[7] T. Idé, S. Papadimitriou, and M. Vlachos, "Computing correlation anomaly scores using stochastic nearest neighbors," in *Proc. 7th IEEE Int. Conf. Data Min (ICDM)*, 2007, pp. 523–528.

[8] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani. (2018). *A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security*. [Online]. Available: https://arxiv.org/abs/1807.11023

[9] N. Cao, S. B. Nasir, S. Sen, and A. Raychowdhury, "Self-optimizing IoT wireless video sensor node with in-situ data analytics and context-driven energy-aware real-time adaptation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 9, pp. 2470–2480, Sep. 2017.

[10] M. Zou, C. Wang, F. Li, and W. Song, "Network phenotyping for network traffic classification and anomaly detection," in *Proc. IEEE Int. Symp. Technol. Homeland Security (HST)*, 2018, pp. 1–8.

[11] T. Nishio, R. Shinkuma, T. Takahashi, and N. B. Mandayam, "Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud," in *Proc. 1st Int. Workshop Mobile Cloud Comput. Netw.*, 2013, pp. 19–26.

[12] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things architecture, possible applications and key challenges," in *Proc. 10th Int. Conf. IEEE Front. Inf. Technol. (FIT)*, 2012, pp. 257–260.

[13] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of Things (IoT) security: Current status, challenges and prospective measures," in *Proc. 10th Int. Conf. IEEE Internet Technol. Secured Trans. (ICITST)*, 2015, pp. 336–341.

[14] M. Valero, F. Li, S. Wang, F.-C. Lin, and W. Song, "Real-time cooperative analytics for ambient noise tomography in sensor networks," *IEEE Trans. Signal Inf. Process. Netw.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8496802. doi: 10.1109/TSIPN.2018.2876751.

[15] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-service detection in 6LoWPAN based Internet of Things," in *Proc. IEEE 9th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, 2013, pp. 600–607.

[16] Y. Elkhatib *et al.*, "On using micro-clouds to deliver the fog," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 8–15, Mar./Apr. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7867723. doi: 10.1109/MIC.2017.35.

[17] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the Internet of Things: A survey of existing protocols and open research issues," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1294–1312, 3rd Quart., 2015.

[18] A. P. R. da Silva *et al.*, "Decentralized intrusion detection in wireless sensor networks," in *Proc. 1st ACM Int. Workshop Qual. Service Security Wireless Mobile Netw.*, 2005, pp. 16–23.

[19] S. Raza, L. Wallgren, and T. Voigt, "SVELTE: Real-time intrusion detection in the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2661–2674, 2013.

[20] D. R. Raymond, R. C. Marchany, M. I. Brownfield, and S. F. Midkiff, "Effects of denial-of-sleep attacks on wireless sensor network MAC protocols," *IEEE Trans. Veh. Technol.*, vol. 58, no. 1, pp. 367–380, Jan. 2009.

[21] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proc. 6th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2005, pp. 46–57.

[22] D. R. Raymond and S. F. Midkiff, "Denial-of-service in wireless sensor networks: Attacks and defenses," *IEEE Pervasive Comput.*, vol. 7, no. 1, pp. 74–81, Jan./Mar. 2008.

[23] N. Sorkunlu, V. Chandola, and A. K. Patra, "Tracking system behavior from resource usage data," in *Proc. IEEE Int. Conf. Clust. Comput. (CLUSTER)*, 2017, pp. 410–418.

[24] L. Caviglione, M. Gaggero, J.-F. Lalande, W. Mazurczyk, and M. Urbański, "Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 799–810, Apr. 2016.

[25] T. Idé and H. Kashima, "Eigenspace-based anomaly detection in computer systems," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2004, pp. 440–449.

[26] Glosser. *Artificial Neural Network, [CC BY-SA 3.0]*. Accessed: Nov. 16, 2018. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=24913461

[27] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.

[28] R. Dey and F. M. Salemt, "Gate-variants of gated recurrent unit (GRU) neural networks," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2017, pp. 1597–1600.

[29] Jeblad. *Gated Recurrent Unit Cell, [CC BY-SA 4.0]*. Accessed: Dec. 27, 2018. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=66225938

[30] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, 2000.

[31] O. A. Grigg, V. T. Farewell, and D. J. Spiegelhalter, "Use of risk-adjusted CUSUM and RSPRT charts for monitoring in medical contexts," *Stat. Methods Med. Res.*, vol. 12, no. 2, pp. 147–170, 2003.

[32] H. A. Abdul-Ghani, D. Konstantas, and M. Mahyoub, "A comprehensive IoT attacks survey based on a building-blocked reference model," *Int. J. Adv. Comput. Sci. Appl. (IJACSA)*, vol. 9, no. 3, pp. 355–373, 2018.

[33] M. M. Ahemd, M. A. Shah, and A. Wahid, "IoT security: A layered approach for attacks & defenses," in *Proc. Int. Conf. IEEE Commun. Technol. (ComTech)*, 2017, pp. 104–110.

[34] *The Five-Layer TCP/IP Model: Description/Attacks/Defense—Computing and Software WiKi*, McMaster Univ., Hamilton, ON, Canada, 2008.

[35] *OWASP Top 10—2013 The Ten Most Critical Web Applications Security Risks*, OWASP Found., Annapolis, MD, USA, 2013.

[36] S. Kumarasamy and A. Gowrishankar, "An active defense mechanism for TCP SYN flooding attacks," *CoRR*, vol. abs/1201.2103, 2012. [Online]. Available: http://arxiv.org/abs/1201.2103

[37] F. Forster and S. Harl. (2012). *Collectd: The System Statistics Collection Daemon*. [Online]. Available: https://collectd.org/

[38] *InfluxDB*. Accessed: Aug. 1, 2018. [Online]. Available: https://www.influxdata.com/

[39] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras

[40] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016. [Online]. Available: https://arxiv.org/abs/1603.04467

[41] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA: Insecure, 2009.

Authors' photographs and biographies not available at the time of publication.