Log-Based Anomaly Detection With Robust Feature Extraction and Online Learning

Shangbin Han, Qianhong Wu, Han Zhang, *Member, IEEE*, Bo Qin, Jiankun Hu, *Senior Member, IEEE*, Xingang Shi, *Member, IEEE*, Linfeng Liu, and Xia Yin

***Abstract*— Cloud technology has brought great convenience to enterprises as well as customers. System logs record notable events and are becoming valuable resources to track and investigate system status. Detecting anomaly from logs as fast as possible can improve the quality of service significantly. Although many machine learning algorithms (e.g., SVM, Logistic Regression) have high detection accuracy, we find that they assume data are clean and might have high training time. Facing these challenges, in this paper, we propose Robust Online Evolving Anomaly Detection (ROEAD) framework which adopts Robust Feature Extractor (RFE) to remove the effects of noise and Online Evolving Anomaly Detection (OEAD) to dynamic update parameters. We propose Online Evolving SVM (OES) algorithm as the example of online anomaly detection methods. We analyze the performance of OES in theory and prove the performance difference between OES and the best hypothesis tends to zero as time goes infinity. We compare the performance of ROEAD against state-of-the-art anomaly detection algorithms using public log datasets. The results demonstrate that ROEAD is able to remove the effects of noise and OES can improve the detection accuracy by more than 40%.**

***Index Terms*— Online learning, ROEAD, RFE.**

Manuscript received July 20, 2020; revised November 19, 2020; accepted January 2, 2021. Date of publication January 21, 2021; date of current version February 12, 2021. This work was supported in part by the National Key Research and Development Program of China under Project 2020YFB10056, Project 2019QY(Y)0602, Project 2017YFB1400700, and Project 2017YFB0802500; in part by the Natural Science Foundation of China under Project 62002009, Project 61932011, Project 61972019, Project 61772538, Project 61532021, Project 91646203, and Project 61672083; and in part by the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, NJUPT. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Eduard A. Jorswieck. *(Corresponding author: Han Zhang.)*

Shangbin Han and Qianhong Wu are with the School of Cyber Science and Technology, Beihang University, Beijing 100191, China, and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China (e-mail: hanshangbin@buaa.edu.cn; qianhong.wu@buaa.edu.cn).

Han Zhang is with the School of Cyber Science and Technology, Beihang University, Beijing 100191, China, and also with the INSC&BNRist, Tsinghua University, Beijing 100084, China (e-mail: zhhan@buaa.edu.cn).

Bo Qin is with the School of Information, Renmin University of China, Beijing 100872, China (e-mail: bo.qin@ruc.edu.cn).

Jiankun Hu is with the School of Engineering and Information Technology, University of New South Wales, Canberra, NSW 2600, Australia (e-mail: j.hu@adfa.edu.au).

Xingang Shi is with the INSC&BNRist, Tsinghua University, Beijing 100084, China.

Linfeng Liu is with the Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China.

Xia Yin is with the DCST, Tsinghua University, Beijing 100084, China.

Digital Object Identifier 10.1109/TIFS.2021.3053371

I. Introduction

NOWADAYS, many companies are migrating business to cloud to obtain good quality-of-service (QoS). System health is becoming one of the primary concerns as any accident will lead to irreparable performance loss. Systems are increasingly more complex and they will generate much more data than ever before. According to the report of Cisco [1], global data center traffic will reach to 20.6 ZB by the end of 2021, while it is only 6.0 ZB in 2016, with an annual increasing rate of 36%. It is extremely difficult to detect anomaly with such huge data volume.

Logs are universally available in nearly all computer systems and they record system states and significant events at various critical points. The noteworthy events of system logs can be used to detect anomalies in a timely manner [8], [16], [24], [25], [36], [37], [45]. For log-based anomaly detection, the unstructured, free-text log entries should be firstly parsed into an *event template* (constant part) with some specific *parameters* (variable part). For example, the raw log message *"Received block blk_822 of size 63"* can be parsed into an event template *"Received block ∗ of size"*, where "∗" denotes the parameters. Then the detection model is used to check whether the coming log entries contain anomalies.

Machine learning technology [10], [40] is a promising tool to dig hidden characters from labeled data. An optimal machine learning algorithm can fast and correctly diagnose the health of system. Up still now, many literatures have proposed to use machine learning technologies (e.g., SVM [6], Logistic Regression [15]) to detect anomalies from system logs. Although the algorithms have high accuracy, they still have the following three **flaws**: Firstly, they assume logs are clean and can't dynamically adapt to noise (e.g., loss, duplication, extra information). In reality, system logs might contain noisy data due to casualness of operators or system bugs. For example, the correct entry is *"Received block blk_822 of size 63"*, but *"Received block blk_822 size 63"* might be logged by operators or programmers. Most detection systems regard the two entries as different ones, but they contain the same noteworthy event (i.e., the size of block blk_822 is 63). Therefore, detection algorithms might gain error inputs and their accuracy can be influenced by this. Secondly, plenty of algorithms need to access the entire training data set to estimate the best parameters which is time-consuming and the parameters can't be adapt to testing data set. Testing data might have different characters as training data set, so that parameters derived

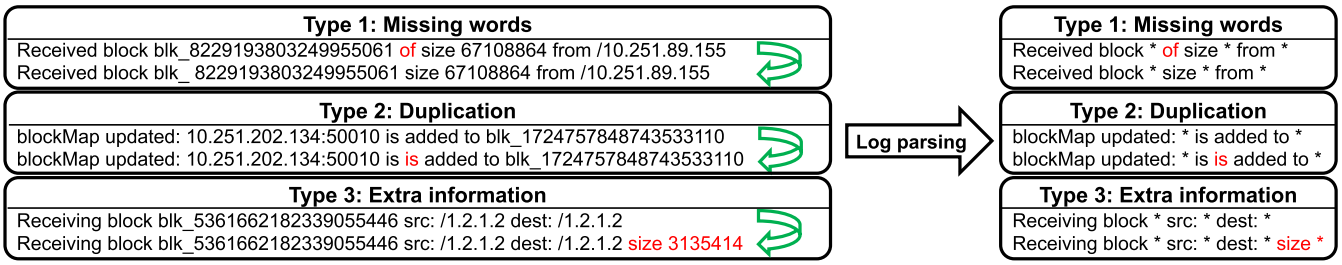


Fig. 1. Logging messages might contain errors due to casualness of operators or system bugs.

from training set can lead to high false alarm rate. Also, training phase might cost much time when data volume is large. Thirdly, many supervised algorithms (e.g., SVM, KNN) need labeled data to train the best parameters, but labels might be impossible to get for system logs. Facing these challenges, in this paper, we make the following **contributions**:

Firstly, we find the deficiency of log-based anomaly detection algorithms. This is because dirty entries may create too much extra events which can affect the accuracy and testing data set might have different characters with the training data set.

Secondly, we propose Robust Online Evolving Anomaly Detection (ROEAD) Framework, which adopts natural language processing to remove the effects of noise and uses online learning theory to dynamic update parameters. We propose Online Evolving SVM (OES) algorithm as the example of online anomaly detection methods. We analyze the performance of OES in theory and prove the performance difference between OES and the best hypothesis tends to zero as time goes infinity.

Thirdly, we compare the performance of ROEAD and OES against state-of-the-art anomaly detection algorithms (e.g., Invariants Mining [28], Log Clustering [26], DeepLog [13]) using three public log datasets (i.e., HDFS, OpenStack, BGL) [23]. The results show that ROEAD is able to remove the effects of noise and OES can improve the detection accuracy by more than 40%.

The remainder of the paper is organized as follows. In Section II, we introduce the background and motivation. Section III presents the design details of ROEAD framework and OES algorithm. We evaluate the performance of ROEAD and OES in Section IV. Section V shows the related works. Section VI concludes this paper.

II. Background and Motivation

Real systems inevitably experience failure. System logs are designed for recording notable events to ease debugging and they are becoming valuable resources to track and investigate system status.

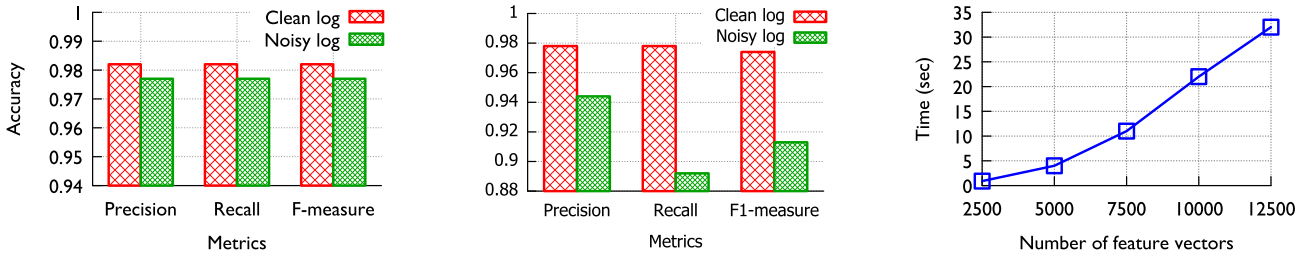
A. Logs are Now Widely Used in Debuging

Failure analysis is often conducted by collecting event logs (or simply logs). Developers and IT operators can write logging statements to record system behaviors to help them debug and maintain systems [18]. For example, in the logging message: *log.error ("Invalid ID:", + ID)*, the statement is at the error level, which is used to record information that may cause system oddities. In reality, there are different verbosity levels (e.g., trace, debug, info, warn, error, and fatal) to record system information. As a result, various messages are generated by these statements which can be used to detect anomalies of systems.

B. Log-Based Anomaly Detection Framework

Log-based Anomaly Detection Framework usually contains *four* parts: Log collection, Log parsing, Feature extraction, Anomaly detection [15]. In modern systems, logs are generated in real time and should be collected in a centralized place under log collection component (e.g., Flume [3]). Then Log parsing component converts the unstructured messages into a map of structured event templates. After this, Feature extraction component forms the feature matrix. At last, Anomaly detection component checks whether a given feature vector is an anomaly or not.

In reality, logging messages might contain errors due to casualness of operators or system bugs and we can regard the errors as **noise**. As Fig. 1 shown, some words or phrases might be missed, duplicated or extra added over the **clean** logs when operators call *log* or *print* methods. Nowadays, although state-of-the-art online log parsing mechanisms (e.g., Drain [21], LogParse [31]) can parse raw logs into log events efficiently, they ignore the log noise, which can lead to high false alarm rate. We now take LogParse [31] as the example and use the public HDFS data set [47] to demonstrate this. We insert 1% extra data into the original HDFS log dataset as noise. Fig. 2(a) illustrates the accuracy of training data set when we use Support Vector Machine (SVM) as the detection algorithm. We can see that the accuracy is high for both clean and noisy logs, i.e., more than 95%. Fig. 2(b) shows the accuracy is high for clean logs but more than 10% decreasing for noisy logs over the testing data set. The reason for this is that log parsing generates too much error events, therefore, many detection errors occur. Besides, parameter estimation technique Maximum Likelihood Estimation (MLE) [35] can be used to derive the best parameters. It has high accuracy but huge computational cost. Fig. 2(c) shows the parameter fitting time with different solving methods on a server with 16 Intel Xeon E5-2630 CPUs, 32G memory. We can see that, even the fastest method will take more than 20s when the scale of training items is larger than 10K. The time is too large for



(a) Training accuracy (b) Testing accuracy (c) Training efficiency

Fig. 2. Log parsing technologies always ignore noise and this can hurt detection accuracy. The offline algorithm takes too much time for best parameters, which hurts the spirit of online detection.

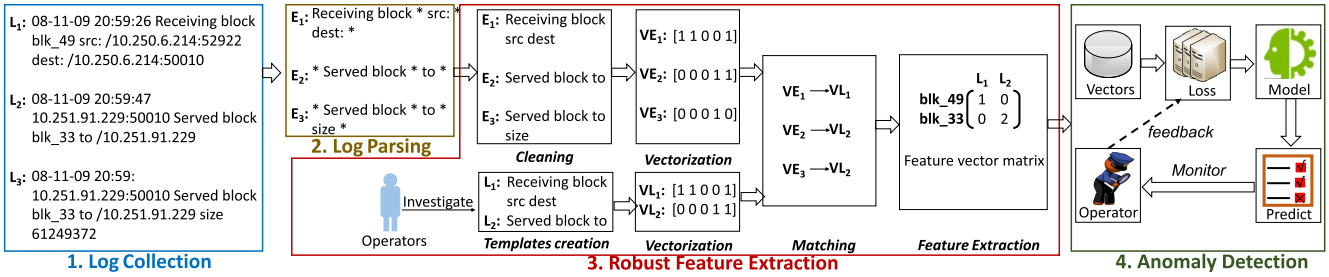


Fig. 3. ROEAD framework contains four parts, i.e., Log collection, log parsing, robust feature extraction and online evolving anomaly detection.

online prediction, which makes it hard to use on the large data set.

III. Robust Online Evolving Anomaly Detection Framework

In this section, we present Robust Online Evolving Anomaly Detection (ROEAD) framework which performs log analysis for large scale systems. As shown in Fig. 3, ROEAD is also composed of four components, i.e., Log collection, Log parsing, Robust feature extraction and Online evolving anomaly detection. As Log collection and Log parsing are similar to pervious literatures (e.g., [13], [28]), we introduce the details in Appendix A. In this part, we just focus on Robust feature extraction and Online evolving anomaly detection.

A. Robust Feature Extraction

After Log parsing, the messages are parsed into events which contains *event template* (constant part) with some specific *parameters* (variable part). Robust Feature Extraction (RFE) aims to extract valuable features from log events that could be fed into anomaly detection models. The key idea of RFE is to use natural language processing to remove the effects of noise. RFE outputs a feature matrix X, whose entry $X[i, j]$ records the i-th group of j-th event template. As Fig. 3 shown, there are three steps to generate this matrix:

1) Step 1: Templates Corpus Creation: IT operators can consider semantics from the perspective of applications to build a corpus of event templates. The event templates contain words with meaningful semantic information and they are stored offline for events correction. As a result, log events with the same semantics can be grouped into one category finally. In the example shown in Fig. 3, two event templates are considered by operators, i.e., L_1 and L_2.

2) Step 2: Events Correction: In reality, logs might contain errors. To reduce the impacts of errors, event templates should be corrected according to their semantic. To achieve this, we address the following three operations: (1) *Cleaning*. Special symbols in parsed log events, such as $*$ and $/$, etc., are cleared by pre-defined regularization rules. Then the compound words are split into independent ones with word segmentation algorithms (e.g., [5]); (2) *Vectorization*. Event templates are mapped to vectors with natural language processing technologies (e.g., Continuous Bag-of-Words [43]). After mapping, we can use vectors to present event templates; (3) *Mactching*. Let $\mathbf{v}_E$ denote the coming event vector and $\mathbf{v}_T$ present one of corpus templates, then Cosine Similarity is used to measure the difference between them:

$$cos(\theta) = \frac{\mathbf{v}_E \cdot \mathbf{v}_T}{||\mathbf{v}_E|| \times ||\mathbf{v}_T||} \tag{1}$$

where $\mathbf{v}_E \cdot \mathbf{v}_T$ denotes the inner product of the two vectors, and $||.||$ presents the size of vector. If the value is larger than a threshold ε (e.g., 95%), the event templates contain the same semantic, otherwise, they are different event templates. For example, E_1, E_2, E_3 map to L_1, L_2 after events correction process.

3) Step 3: Feature Extraction: In order to extract features, we firstly need to separate log data into various groups according to their identifiers. Logs may adopt different identifiers, for example, OpenStack logs use instance_id as identifiers since they record the status of VM instances, and HDFS logs use block_id as identifiers. Then, we count the occurrence number of each log event in each group and finally get the feature vector matrix. In the example shown in Fig. 3, event L_2 occurs twice in blk_33 and event L_1 occurs once in blk_49, then we derive the feature vector matrix at last.

B. Online Evolving Anomaly Detection Algorithm

Each row $\mathbf{x} \in X$ records the feature of the detection entry. Let $\mathbf{x}_t$ denote its value in t and the reported value $y_t \in \{-1, 1\}$ can be derived by an anomaly detection model $Y(\mathbf{x}_t, \mathbf{w}_t)$, whose input parameters are feature measured vector $\mathbf{x}_t$ and feature importance vector $\mathbf{w}_t$. We use the model to check whether the corresponding entry contains anomalies (i.e., $y_t = 1$) or not (i.e., $y_t = -1$).

As system evolves, static weight vectors might cause much error since they can't adapt fluctuation. Therefore, $\mathbf{w}_t$ should be smart enough to *dynamically adjust* according to environment. To achieve this, we consider to use the online learning technology [20] and incorporate the feedback of IT operators into anomaly detection algorithms. As shown in Fig. 3, IT operators firstly derive the state of system (i.e., y_t) with parameters $\mathbf{x}_t$ and $\mathbf{w}_t$. Then the true result (i.e., $\hat{y}_t$) is observed by the IT operators. If the true result doesn't match the reported result (i.e., $\hat{y}_t \neq y_t$), a loss $f_t(\mathbf{w}_t)$ will occur. At last, the loss will return to anomaly detection algorithm to help amending $\mathbf{w}_t$.

The framework has the following advantages: Firstly, it doesn't require specific parameter training phase, while most traditional data-driven anomaly detection algorithms need complex and time-consuming training phase to derive the best parameters. Secondly, it can adapt to system fluctuation but most anomaly detection algorithms only perform well on training data set whose characters might be different from the whole data set. Thirdly, its performance doesn't rely on specific algorithms and plenty of learning methods (e.g., SVM, KNN, DT) fit the framework. Fourthly, it uses feedback of IT operators to adjust parameters rather than the labels of training data sets.

C. Online Evolving SVM

Support Vector Machine (SVM) is a machine learning algorithm which is able to establish an optimal decision hyperplane to distinguish categories. It is an offline algorithm. We now borrow the idea of SVM and propose the Online Evolving SVM (OES) model to detect anomaly at time slot t. Give the feature importance vector $\mathbf{w}_t$ and feature vector $\mathbf{x}_t$, we can use them to detect anomaly:

$$y_t = sign(\mathbf{w}_t^\top \mathbf{x}_t) \tag{2}$$

where $sign(x) \in \{-1, 1\}$ is the sign function. We define the candidate convex set for $\mathbf{w}_t$, $\forall t \in T$:

$$S = \left\{ \mathbf{w}_t \middle| \mathbf{w}_t \in \mathbb{R}^{|J|} : ||\mathbf{w}_{t,j}|| \leq L, j \in J \right\} \tag{3}$$

where $\mathbf{w}_{t,j}$ is the j-th element of vector $\mathbf{w}_t$ and total J event templates are considered. Mathematically, we should try to seek $\mathbf{w}$ that minimizes the incorrectly classified examples over the total T iterations, i.e.,

$$\min_{\mathbf{w} \in S} \sum_{t \in T} \theta(sign(\mathbf{w}_t^\top \mathbf{x}) \neq \hat{y}_t) \tag{4}$$

where $\theta(z) \in \{0, 1\}$ is the indicator function that takes the value 1 if z is satisfied and 0 otherwise. $\hat{y}_t = \pm 1$ is the true label at t. However, (4) is hard to use in practice since it is 0/1 loss and needs to access the whole data set. In this case, we would like to replace it with a convex loss function at each iteration t:

$$f_t(\mathbf{w}_t) = \max\{0, 1 - \hat{y}_t \cdot \mathbf{w}_t^\top \mathbf{x}_t\} \tag{5}$$

To derive the best feature vector $\mathbf{w}_t$ by standing at current round t, we can use any vector which has minimal loss on all past rounds:

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \sum_{i=1}^{t} f_i(\mathbf{w}) + R(\mathbf{w}) \tag{6}$$

where $R(\mathbf{w})$ is a regularized function (i.e., the regularizer) to stabilize the prediction. From (6), we can derive the best parameters. However, it is often time-consuming to solve, which hurts the spirits of online detection. In this case, we adopt the online mirror descent algorithm [30] and change (5) to its gradient function, i.e.,

$$f_t(\mathbf{w}_t) \simeq \bigtriangledown f_t(\mathbf{w}_t)\mathbf{w}_t \tag{7}$$

where $\bigtriangledown f_t(\mathbf{w}_t)$ is the gradient of f_t at $\mathbf{w}_t$, which can be derived as:

$$\bigtriangledown f_t(\mathbf{w}_t) = \begin{cases} -\hat{y}_t \cdot \mathbf{x}_t & \hat{y}_t \neq y_t \\ 0 & \hat{y}_t = y_t \end{cases} \tag{8}$$

The regularization term $R(\mathbf{w})$ is defined as follows,

$$R(\mathbf{w}) = \frac{1}{\delta}\|\mathbf{w}\|_1 + \frac{1}{2}\sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right)\|\mathbf{w} - \mathbf{w}_i\|_2^2 \tag{9}$$

where the first term is L_1-regularization that aims to obtain sparse solution [30] and δ is a constant. The second term L_2-regularization is to ensure the choice stability of $\mathbf{w}$, and η_t is the learning rate. Take (7) and (9) into (6), we can derive a convex optimization problem with no constraints. According to KKT (Karush–Kuhn–Tucker) condition [11], we take the derivative of (6) equals to 0. The equation is easy to be solved (shown in Appendix B) and we can get the optimal feature weight vector finally:

$$\mathbf{w}_{t+1,j} = \begin{cases} -\eta_t \left[\gamma_{t,j} - sign(\gamma_{t,j}) \cdot \frac{1}{\delta}\right] & |\gamma_{t,j}| > \frac{1}{\delta} \\ 0 & |\gamma_{t,j}| \leq \frac{1}{\delta} \end{cases} \tag{10}$$

where $\gamma_{t,j}$ is defined as follows:

$$\gamma_{t,j} = \sum_{i=1}^{t}\left[\bigtriangledown f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right] \tag{11}$$

Finally, Algorithm 1 shows the details of applying OES to anomaly detection. Algorithm 1 takes event templates set J and δ as its input. After receiving a feature vector $\mathbf{x}$ (Line 3), it firstly derives the feature importance vector $\mathbf{w}$ according to (10) (Line 4-5). Then it uses the vector to detect anomaly (Line 6). Next the real system health is returned from IT operators (Line 7). At last, parameters are updated with the detection loss (Line 8-12).

Algorithm 1: Online Evolving SVM (OES)

Input: δ, J

$\gamma_j = 0, \forall j \in J$;
for $t \in T$ **do**
  Receive feature vector $\mathbf{x}_t$;
  for $j \in J$ **do**
    $$\mathbf{w}_{t,j} = \begin{cases} -\eta_t \left[\gamma_j - sgn(\gamma_j) \cdot \frac{1}{\delta}\right] & |\gamma_{t,j}| > \frac{1}{\delta} \\ 0 & |\gamma_{t,j}| \leq \frac{1}{\delta} \end{cases}$$
  Anomaly detection with $y = sign(\mathbf{w}_t^\top \mathbf{x}_t)$;
  Get feedback $\hat{y}_t \in \{+1, -1\}$ from IT operators;
  if $\hat{y}_t == y_t$ **then**
    $\bigtriangledown f_t(\mathbf{w}_t) = 0$;
  else
    $\bigtriangledown f_t(\mathbf{w}_t) = -y_t * x_j$;
  $\gamma_j = \gamma_j + (\bigtriangledown f_t(\mathbf{w}_{t,j}) - (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}})\mathbf{w}_{t,j}), \forall j \in J$;

1) Analysis: The **regret** metric [20] is always used to measure the difference between the online algorithm and the best static decision in hindsight. We are interested in the upper bound on the worst case regret of an algorithm. A good online learning algorithm is one with a regret (defined below) that grows sub-linearly as a function of T. This implies that on average the algorithm is performing as the best static strategy we can have in hindsight,. i.e.,

$$Regret_T = \max_{\mathbf{w}^* \in S} \left\{ \sum_{t=1}^{T} f_t(\mathbf{w}_t) - \sum_{t=1}^{T} f_t(\mathbf{w}^*) \right\} = o(T) \quad (12)$$

To derive the performance bound on OES, we begin with the following lemma.

Lemma 1: Compared with the best static decision $\mathbf{w}^*$, the *Regret* upper bound satisfies:

$$Regret_T \leq \frac{1}{2} \sum_{t=1}^{T} (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) \|\mathbf{w}^* - \mathbf{w}_t\|^2 + \frac{1}{2} \sum_{t=1}^{T} \eta_t \bigtriangledown f_t^2(\mathbf{w}_t) \quad (13)$$

The proof can be derived just as a similar procedure to show the ratio in [30] and therefore is omitted. With Lemma 1, we can prove the following regret bound.

Theorem 1: Let G denote the maximal value of feature vector entry. When we set $\eta_t = \frac{L}{G\sqrt{t}}$, we have:

$$Regret_T \leq 3JGL\sqrt{T} \quad (14)$$

Proof: $\forall \mathbf{u}, \mathbf{v} \in S, \|\mathbf{u} - \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$. According to (3), we have $\|\mathbf{u}\| \leq \sqrt{JL}$, thus, $\|\mathbf{u} - \mathbf{v}\| \leq 2\sqrt{JL}$. Note that $\bigtriangledown f_t(\mathbf{w}_t) \in \{-\hat{y}_t \cdot \mathbf{x}_t, 0\}$ and $||\mathbf{x}_t|| \leq \sqrt{JG}$, so we have $\bigtriangledown f_t(\mathbf{w}_t) \leq \sqrt{JG}$. Since $\eta_t = \frac{L}{G\sqrt{t}}$ (with $\frac{1}{\eta_0} \triangleq 0$), then according to Lemma 1:

$$\begin{aligned} Regret_T &\leq \frac{1}{2} \sum_{t=1}^{T} (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) \|\mathbf{w}^* - \mathbf{w}_t\|^2 + \frac{1}{2} \sum_{t=1}^{T} \eta_t \bigtriangledown f_t^2 \\ &\leq 2JL^2 \sum_{t=1}^{T} (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}) + \frac{1}{2} JG^2 \sum_{t=1}^{T} \eta_t \\ &= 2JL^2 \frac{1}{\eta_T} + \frac{1}{2} JG^2 \sum_{t=1}^{T} \eta_t \\ &= 2JGL\sqrt{T} + \frac{1}{2} JGL \sum_{t=1}^{T} \frac{1}{\sqrt{t}} \\ &\leq 3JGL\sqrt{T} \end{aligned} \quad (15)$$

The last inequality follows since $\sum_{t=1}^{T} \frac{1}{\sqrt{t}} \leq 2\sqrt{T}$. ■

Theorem 1 shows the difference between OES and the best hypothesis tends to zero as T goes to infinity.

IV. Evaluation

In this section, we thoroughly evaluate the performance of ROEAD and OES with public log datasets and corresponding synthetic noise log datasets. The core code is implemented in Python language. Source codes can be downloaded in [4]. Main results of our experiments can be concluded as:

- Compared with state-of-the-art anomaly detection algorithms, OES performs about 5%, 6%, 10%, 15%, 20% better than LR, SVM, Deeplog, IM, LC, respectively.
- ROEAD can remove the effects of noise. Large scale comparisons show that ROEAD performs about 14.1%, 15.5%, 13.9%, 17.5%, 20.8% better than SVM, LR, Deeplog, IM and LC, respectively under logs with noise.
- ROEAD is a framework and doesn't rely on any particular method. It has high detection accuracy and efficiency. Other ROEAD methods have comparable accuracy.

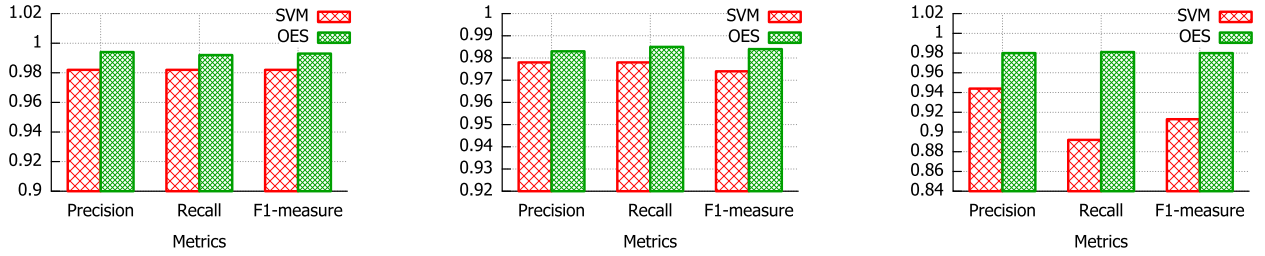
A. Evaluation Methodology

1) Data Set: Three public log datasets [23] are used in our evaluation:

a) HDFS Log Data Set: It is generated through running Hadoop-based map-reduce jobs on 203 Amazon's EC2 nodes [47], and labeled by Hadoop domain experts. Among 11,197, 954 log entries being collected, about 2.9% are abnormal and we consider 29 events in total.

b) OpenStack Log Data Set: It is generated through an OpenStack experiment (version Mitaka) deployed CloudLab [2] with one control node, one network node and eight compute nodes. Among 1,335,318 log entries collected, about 7% are abnormal. A script was running to constantly execute VM-related tasks, including VM creation/deletion, stop/start, pause/unpause and suspend/resume. We consider 675 events in total.

c) BGL Log Data Set: It is generated from BlueGene/L supercomputer system deployed at Lawrence Livermore National Labs (LLNL) [39]. The log has two labels, where normal messages are indicated by "-". Among the total 200,000 log entries are considered, about 1.5% logs are abnormal and we consider 139 events in total.



(a) Training under clean log (b) Testing under clean log (c) Testing under noisy log

Fig. 4. OES outperforms SVM under both clean and noisy logs.

2) Setup: We conduct all our experiments on a Linux server with 16 Intel Xeon E5-2630 CPUs, 32G memory and 9T hard disk. The default algorithm in ROEAD is OES. Proper G, L and δ for OES should be set in advance. G is defined as the maximal value of feature vector items. According to log parsing of the public log datasets (e.g., HDFS), we find the number of event templates are smaller than 40 and we use it as the default value of G. L defines the domain of feature importance and its default value is 4 in our experiments. δ is usually selected through cross validation of accuracy and sparsity, the same as the previous literature [44], we set 1000 as its default value. In the following experiments, we sometimes compare the performance of ROEAD with some supervised machine learning algorithms. We randomly choose 80% of each dataset as the training data, and the remaining 20% as testing data. ROEAD uses the feedbacks of IT operators to adjust parameters and we use the labels of public log datasets as the feedback signals.

3) Comparison: We compare the performance of ROEAD with five methods: two classical supervised learning methods Logistic Regression (LR) and SVM, two unsupervised learning approaches Invariants Mining (IM) [28] and Log Clustering (LC) [26], one deep learning method DeepLog [13]. The parameters of these methods are all set best for accuracy.

a) Precision: It refers to the proportion of predicted positive samples that are actually positive samples and is defined as follows: Precision $= \frac{TP}{TP+FP}$, where TP is True Positives and FP is False Positives.

b) Recall: It measures the ability of a model to recognize positive samples and is given as follows: Recall $= \frac{TP}{TP+FN}$, where FN is False Negatives.

c) F1-measure: It takes recall and precision into consideration, which is the harmonic average of the two: F1-measure $= \frac{2*Precision*Recall}{Precision+Recall}$.

B. Revisiting the Motivating Example

In this section, we will revisiting the motivating example to verify the effectiveness of OES. All the parameters are set with default value. We also paint the results of SVM to make comparisons.

Fig. 4(a) and Fig. 4(b) show the training and testing accuracy under clean logs, respectively. We can see that OES performs about 10% better than the SVM. This shows the superiority of online parameters adjustment of OES. Fig. 4(c) shows the testing accuracy under noisy data using the trained model under clean logs. We can see that ROEAD with OES has significant improvement over precision, recall and F1-measure. The reason for this is that ROEAD is able to classify extra events generated in the noisy data. To prove this, Fig. 5 demonstrates the result of Robust feature extraction, where X-axis represents the parsed event templates of noisy log (both normal events and noisy events) and Y-axis represents the templates corpus created by IT operators. The Z-axis is the cosine similarity value between the event templates and templates corpus. The similarity threshold ε is 0.852. We can see that RFE is able to remove the effects of noise. For example, in the red rectangle, four event templates (normal event 6 and noisy events 7, 8, 9) can all inject to corpus event template 4 finally.

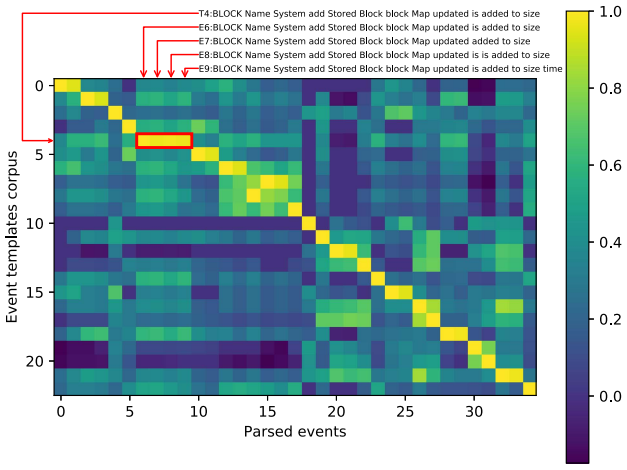
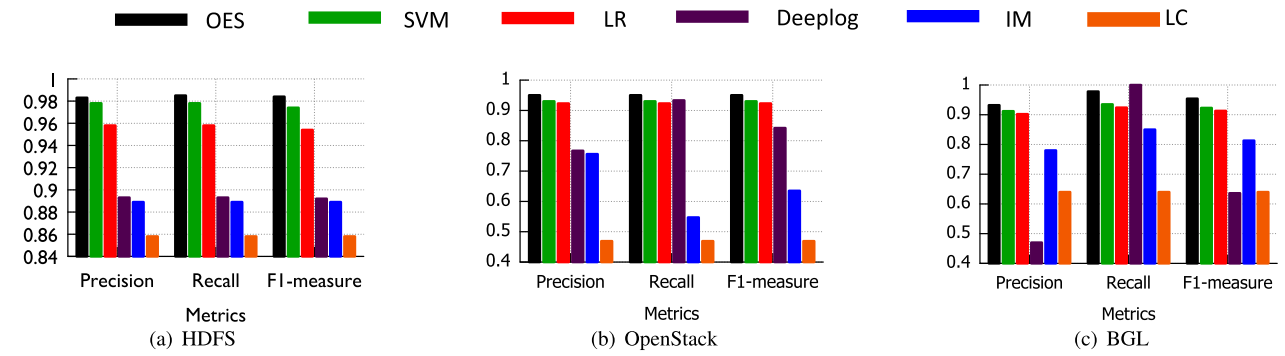


Fig. 5. RFE is able to remove the effects of noise.

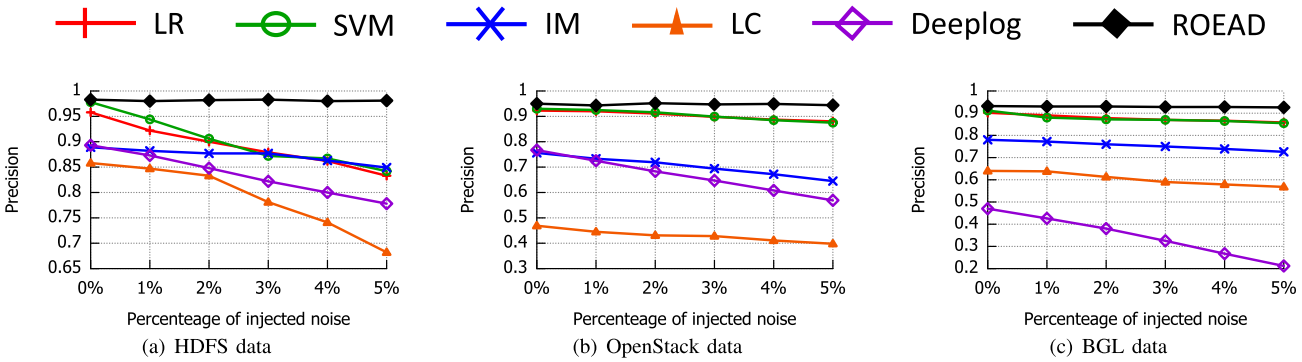
C. Large Scale Comparisons

In this part, we compare the performance of OES with state-of-the-art anomaly detection algorithms. Fig. 6 shows the results on clean HDFS log set, OpenStack log set and BGL log set. We can see that OES achieves the best accuracy. Fig. 6(a) shows the performance under HDFS log set. We can see that OES performs about 5%, 6%, 10%, 15%, 20% better than LR, SVM, Deeplog, IM, LC, respectively. The reason for this is that OES can fully utilize the feedback of IT operators to derive the best parameters. Also, supervised learning methods, including LR, SVM perform better than unsupervised learning approaches, e.g., IM, LC and Deeplog, since they can gain



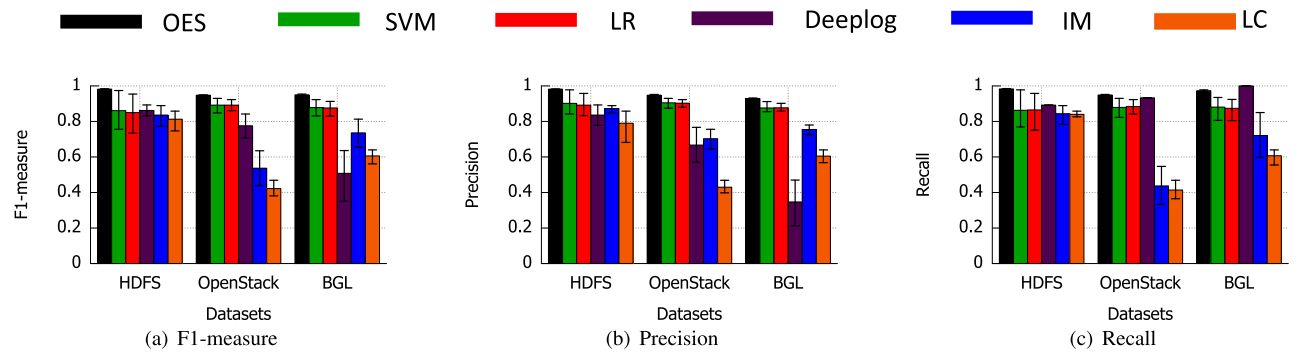
(a) HDFS (b) OpenStack (c) BGL

Fig. 6. Accuracy comparison under different clean log sets for different algorithms.



(a) HDFS data (b) OpenStack data (c) BGL data

Fig. 7. Precision comparison when we inject x% percentage noise into clean logs. ROEAD can remove the effects of noise.



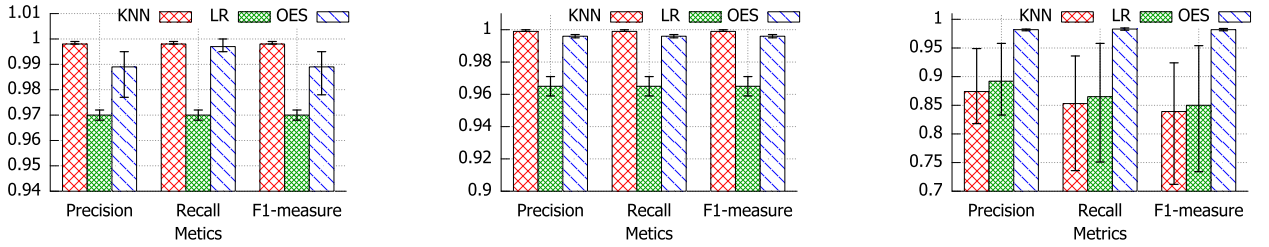
(a) F1-measure (b) Precision (c) Recall

Fig. 8. Accuracy comparison when we inject x% noise data into clean logs, where x% is chosen between 1% and 5% with uniform distribution. All schemes use RFE to remove the effects of noise.

better parameters with the help of training data set labels. Fig. 6(b) shows the performance under OpenStack log set. The result is similar, where OES improves detection accuracy by up to 40%. Fig. 6(c) shows the performance over BGL log set. We can see that although Deeplog has best recall, OES outperforms it for the other two metrics. OES has better performance than other machine learning algorithms and it doesn't need labels to train parameters.

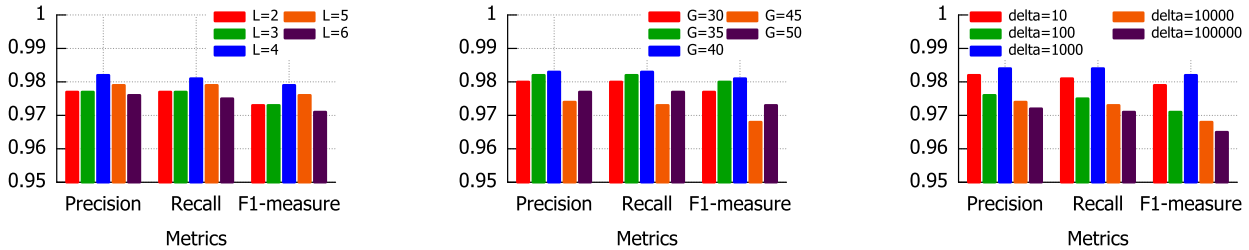
ROEAD is able to remove the effects of noise. We now test the performance of ROEAD by injecting different proportions of noise into clean logs. Fig. 7 shows the comparison results on the HDFS dataset, OpenStack dataset and BGL dataset. Overall, we can see that ROEAD is much more robust with the increase of noise ratio, which shows high accuracy and stable performance. Even at the noise ratio of 5%, ROEAD can still keep the F1-measure over 0.9 under all datasets. However, the performance of the other five methods degrade when noise ratio increases, since the injected noisy log generates new event templates and change the feature matrix dimension. However, ROEAD considers event templates similarity to avoid performance degradation.

Fig. 8 shows the average F1-measure, precision and recall for the three log sets when we inject x% percentage noise data into clean logs, where x% is chosen between 1% and 5% with uniform distribution. We repeat the process 30 times where the error bar paints the maximum and minimum value and all schemes use RFE to filter noise. Fig. 8(a) shows OES is about 14.1%, 15.5%, 13.9%, 17.5%, 20.8% better than SVM, LR, Deeplog, IM and LC for average precision, respectively. The reason for this is that OES can adapt to testing data set.



(a) Training under clean HDFS log data set (b) Testing under clean HDFS log data set (c) Testing under noisy HDFS log data set

Fig. 9. Comparison between OES and supervised learning algorithms.



(a) Different L (b) Different G (c) Different δ

Fig. 10. Performance under different parameters.

Fig. 8(b) shows the average recall. We can see that OES has as large recall as Deeplog. Fig. 8(c) shows that F1-measure of OES is over 95% on average, which is at least 10% better than other schemes.

From the above experiments, we can see that the supervised machine learning algorithms always have higher accuracy than other baselines. We further compare the performance of OES with the supervised learning methods. Fig. 9(a) shows the accuracy under clean HDFS training data set and we can see OES performs at least 10% better than other schemes. Fig. 9(b) shows the accuracy under clean HDFS testing data set and the results are similar to training data set. Fig. 9(c) shows the accuracy under noisy HDFS testing data set and we can see that odds of OES are larger, i.e., more than 20%, since ROEAD is able to handle data noise.

D. Performance Under Different Settings

We now test the performance of ROEAD and OES under different parameters with HDFS dataset.

Three main parameters (i.e., L, G, δ) play important roles in OES. According to (3), L limits the range of weight vector. Fig. 10(a) shows the accuracy comparison under different L. We can see that overall performance is stable. G is defined as the maximal value of feature vector entry, and it depends on different data sets and feature vectors. Fig. 10(b) shows the accuracy comparison under different G. We can see that the accuracy at first increases and then decreases. The reason for this is that small G might result in the importance vector fluctuation since the learning rate might be too large, while large G can make the vector converge slowly. Fig. 10(c) shows the accuracy comparison under different δ. According to (10), large δ will lead to much zero in weight vector $\mathbf{w}$ but can provide nontrivial sparsity. We can see that 1000 is good enough to ensure accuracy finally.

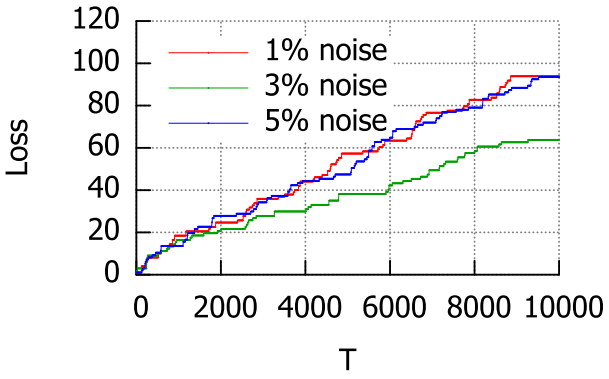


Fig. 11. Performance loss of OES.

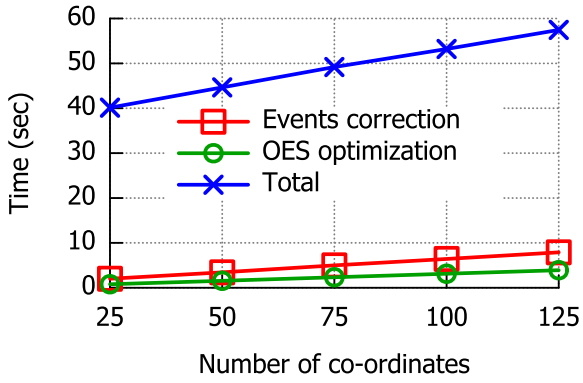
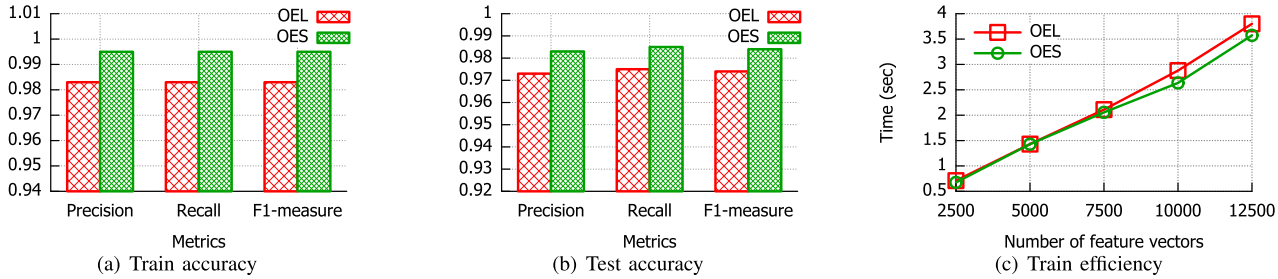


Fig. 12. Cost of ROEAD.

We now evaluate the performance loss of ROEAD under different percentage of noise data. Fig. 11 shows the results, where the X-axis denotes the feedback times from IT operators and Y-axis represents the accumulated loss. We can see that each curve grows slowly with the increase of iteration time T. It is clear that after many iterations (e.g., 9000), the ratio between accumulated loss and iteration time is close to 0.



(a) Train accuracy (b) Test accuracy (c) Train efficiency

Fig. 13. ROEAD is a framework and doesn't rely on particular algorithm.

TABLE I
COMPARISON OF THE MOST RELATED WORKS

Methods	Technique	Advantages	Limitations
[32]	LSTM	Detecting sequential and quantitative anomalies simultaneously	Requiring a lot of storage and bandwidth to calculate
[22]	Clustering	Saving clustering time while maintaining accuracy	Clustering accuracy decreases when data increases
[14]	SVM	Achieving the optimal feature space weighting	Experiments are not rich enough to support the argument
[6]	SVM	Feature selection has a great influence on detection accuracy	The effect of data noise is not considered
[17]	DT	Configurations of performance and time overhead	No consideration about the versatility of scheme design
[15]	Regression	Improved systems reliability	Poor real-time performance
[7]	KNN, SVM, DT	Reduced feature dimension	Inefficient when handling massive amounts of data

As introduced above, ROEAD framework contains four parts. We now explore the cost of Log parsing, Robust feature extraction and Online detection when we inject 1% noise into HDFS dataset. Fig. 12 shows the total, RFE component and online detection time under different size of feature matrix coordinates (i.e., events corpus). We can see that events correction with NLP and OES only occupy 10% and 5% of the total time. Log parsing and Feature extraction are the time consuming components.

We use OES algorithm above as the default online detection algorithm. Indeed, ROEAD is a framework and doesn't rely on particular method. We apply the idea to Logistic Regression and propose Online Evolving Logistic (OEL). Injecting 1% noisy into HDFS clean data set and Fig. 13 shows the results. We can see the two methods have comparable accuracy.

V. RELATED WORKS

Mainly designed for recording system runtime information, logs which contain valuable resources are widely used for various tasks, such as debuging [29], [34], diagnosis [48] and anomaly detection [27], [32], [38].

Log parsing is the first step in analyzing logs, which are broadly studied in recent years. Zhu *et al.* [50] conduct solid experiments to investigate 13 log parsers on various datasets, including online methods Drain [21], Spell [12] and other offline methods MoLFI [33], LogMine [19], etc. Among all the testing methods, Drain shows the best average performance in both accuracy and efficiency. Recently, Nedelkoski *et al.* propose NuLog [38], which utilizes a self-supervised learning model and formulates the parsing as masked language modeling. Meng *et al.* propose LogParse [31], which treats parsing task as word classification problem to learn and update log templates adaptively. Although they show good performance in accuracy and efficiency, they do not consider the error caused by noisy.

Unsupervised learning methods are widely used to detect anomaly. Xu *et al.* [47] employ PCA technique to generate abnormal and normal space by finding patterns from dimensions of event count vectors. Lou *et al.* [28] propose invariant mining (IM) according to the linear relationship between log events in a program execution flow. If a new instance deviates from the invariants, it is an anomaly. Lin *et al.* [26] propose Log Clustering (LC), which uses clustering based technique to classify log events of online system. Vaarandi *et al.* [41] also propose clustering based data mining for detecting anomaly in system log. They are widely used in real world due to the advantage of not requiring manual labels. However, they can be trapped in the local minima.

Deep learning methods are becoming popular to detect anomaly for system log. Du *et al.* propose Deeplog [13], which uses deep neural network LSTM to automatically learn log patterns from normal execution of a system. An anomaly is alarmed if new log patterns deviate from the trained normal patterns. Meng *et al.* [32] and Zhang *et al.* [49] also utilize LSTM to train models for anomaly detection. Nedelkoski *et al.* [38] employ attention-based encoder model to distinguish normal and anomaly logs. These methods have strong learning ability, but the learning process is a black box with non-explanatory, and has high hardware requirements.

Supervised learning models are easy to interpret and have high testing accuracy, so that there is a rising trend of adopting them for anomaly detection. Reference [14] uses robust SVM algorithm for intrusion detection in information systems. References [9] and [17] use Decision Tree (DT) to classify normal and anomalous events. Reference [15] presents a regression-based analysis method to mine the anomalies. Reference [7] tests the performance of supervised machine

learning (e.g., KNN, SVM, DT) on attacks identifying. There are also some hybrid supervised methods [42], [46]. TABLE I shows the comparisons of the most related works.

VI. Conclusion

In this paper, we proposed a ROEAD framework, which contains Log collection, Log parsing, Robust feature extraction and Online evolving anomaly detection. ROEAD adopts Robust Feature Extractor (RFE) to remove the effects of noise and Online Evolving Anomaly Detection (OEAD) to dynamic update parameters. We presented an Online Evolving SVM (OES) algorithm as the example of online anomaly detection methods. We analyzed the performance of OES in theory and proved the performance difference between OES and the best hypothesis tends to zero as time goes infinity. The analysis show ROEAD and OES have good performance under public log datasets.

Appendix A
Log-Based Anomaly Detection Framework

Log-based Anomaly Detection Framework contains four components: Log collection, Log parsing, Feature extraction and Anomaly detection. We introduce Log collection and Log parsing in details.

A. Log Collection

Logs can be generated in network equipment, service applications and systems during operation. Each line of logs records events that occur during the execution of the system, which contains the detailed description of the date, time, action, and so on. These logs provide traces that can be used to understand the activities of the system and diagnose problems, which is a treasure trove of valuable information for anomaly detection. Logs can be collected with a data streaming pipeline, such as Flume [3] which is a distributed, reliable and highly available massive log aggregation system. *Flume Event* is defined as a byte data stream with payload and optional string attribute set. When a log enters Flume, it will be converted to *Flume Event*, which is essentially a JSON format string. The technique structure of Flume has three components: Source, Channel and Sink. These three parts are collectively referred to as the *Flume Agent*, which carries the flow of events from an external source to the next destination. Therefore, Source receives the external *Flume Events*, Channel is the data transmission pipeline and Sink represents the storage location of external data.

B. Log Parsing

The log pattern varies in different systems. Developers use output statements to record runtime variable values, understand the running status and even output complete statement messages for developers or engineers to read and monitor. As a result, the logs are unstructured with free text. To further analyze the valuable data, log parsing is the first key step, which aims to parse unstructured log messages into structured event streams. Take the HDFS log message "PacketResponder 2 for block blk 8229193803249955061 terminating" as the example, it is full of various text and parameters. The text strings show the event template and the parameters record the runtime information, which might change any time. What log parsing does is to change the constants and variables of log message to the event templates (e.g., "PacketResponder $*$ for block $*$ terminating", where "$*$" represents the variable).

Appendix B
Proof of Eq.(10)

Proof: Let $l_{t+1,j}(\mathbf{w}_{t+1,j}) = \sum_{i=1}^{t} \nabla f_i(\mathbf{w}_{i,j})\mathbf{w}_{t+1,j} + R(\mathbf{w}_{t+1,j})$, which is decomposable to the j-th coordinate and the minimizer of each is,

$$\begin{aligned}\mathbf{w}_{t+1,j} &= \arg\min_{\mathbf{w}_{t+1,j}} l_{t+1,j}(\mathbf{w}_{t+1,j}) \\ &= \arg\min_{\mathbf{w}_{t+1,j}} \sum_{i=1}^{t} \left(\nabla f_i(\mathbf{w}_{i,j})\mathbf{w}_{t+1,j}\right) \\ &\quad + \frac{1}{2}\sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right)(\mathbf{w}_{t+1,j} - \mathbf{w}_{i,j})^2 \\ &\quad + \frac{1}{\delta}\left|\mathbf{w}_{t+1,j}\right| \end{aligned} \tag{16}$$

By merging similar items of $l_{t+1,j}(\mathbf{w}_{t+1,j})$, we can get,

$$\begin{aligned} l_{t+1,j}(\mathbf{w}_{t+1,j}) &= \sum_{i=1}^{t}\left[\nabla f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right]\cdot \mathbf{w}_{t+1,j} \\ &\quad + \frac{1}{2}\sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right)\mathbf{w}_{t+1,j}^2 \\ &\quad + \frac{1}{2}\sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right)\mathbf{w}_{i,j}^2 \\ &\quad + \frac{1}{\delta}\left|\mathbf{w}_{t+1,j}\right| \end{aligned} \tag{17}$$

Next, we discuss the analytical solution of $\mathbf{w}_{t+1,j}$ by cases.

- **Case 1:** $\mathbf{w}_{t+1,j} > 0$.
  In this case, the derivative of $l_{t+1,j}(\mathbf{w}_{t+1,j})$ with respect to $\mathbf{w}_{t+1,j}$ is,

$$\begin{aligned}\frac{\partial l_{t+1,j}(\mathbf{w}_{t+1,j})}{\partial \mathbf{w}_{t+1,j}} &= \sum_{i=1}^{t}\left[\nabla f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right] \\ &\quad + \sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right)\mathbf{w}_{t+1,j} \\ &\quad + \frac{1}{\delta}\end{aligned} \tag{18}$$

  where $\sum_{i=1}^{t}\left(\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}}\right) = \frac{1}{\eta_t}$. Let Eq.(18) be 0, and we get the solution,

$$\mathbf{w}_{t+1,j} = -\eta_t\left\{\sum_{i=1}^{t}\left[\nabla f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right] + \frac{1}{\delta}\right\} \tag{19}$$

  Let

$$\gamma_{t,j} = \sum_{i=1}^{t}\left[\nabla f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right].$$

Since $\mathbf{w}_{t+1,j} > 0$ and $\eta_t > 0$, then, $\gamma_{t,j} + \frac{1}{\delta} < 0$. So, $|\gamma_{t,j}| > \frac{1}{\delta}$. As a result, we have,

$$\mathbf{w}_{t+1,j} = -\eta_t \left[\gamma_{t,j} - sgn(\gamma_{t,j}) \cdot \tfrac{1}{\delta}\right], if \ |\gamma_{t,j}| > \tfrac{1}{\delta} \quad (20)$$

- **Case 2:** $\mathbf{w}_{t+1,j} < 0$.
  Following the similar arguments used in case 1, we can get,

$$\mathbf{w}_{t+1,j} = -\eta_t \left[\gamma_{t,j} - sgn(\gamma_{t,j}) \cdot \tfrac{1}{\delta}\right], if \ |\gamma_{t,j}| > \tfrac{1}{\delta} \quad (21)$$

- **Case 3:** $\mathbf{w}_{t+1,j} = 0$.
  Obviously, when $|\gamma_{t,j}| > \frac{1}{\delta}$, $\mathbf{w}_{t+1,j} = 0$.

Combining the above three situations, we finally get the solution,

$$\mathbf{w}_{t+1,j} = \begin{cases} -\eta_t \left[\gamma_{t,j} - sgn(\gamma_{t,j}) \cdot \frac{1}{\delta}\right] & |\gamma_{t,j}| > \frac{1}{\delta} \\ 0 & |\gamma_{t,j}| \leq \frac{1}{\delta}. \end{cases} \quad (22)$$

where

$$\gamma_{t,j} = \sum_{i=1}^{t} \left[\nabla f_i(\mathbf{w}_{i,j}) - (\frac{1}{\eta_i} - \frac{1}{\eta_{i-1}})\mathbf{w}_{i,j}\right].$$

■

REFERENCES

[1] *Cisco Global Cloud Index: Forecast and Methodology*, Cisco, San Jose, CA, USA, 2016.

[2] (Jan. 2021). *Cloudlab*. [Online]. Available: https://cloudlab.us/

[3] (Jan. 2021). *Flume*. [Online]. Available: http://flume.apache.org/

[4] (Jan. 2021). *Roead*. [Online]. Available: https://github.com/JasonHans/ROEAD-core-code

[5] (Jan. 2021). *Splitter*. [Online]. Available: https://pypi.org/project/compound-word-splitter/

[6] D. Aksu and M. Ali Aydin, "Detecting port scan attempts with comparative analysis of deep learning and support vector machine algorithms," in *Proc. Int. Congr. Big Data, Deep Learn. Fighting Cyber Terrorism (IBIGDELFT)*, Dec. 2018, pp. 77–80.

[7] D. Aksu, S. Üstebay, M. A. Aydin, and T. Atmaca, "Intrusion detection with comparative analysis of supervised learning techniques and Fisher score feature selection algorithm," in *Proc. Int. Symp. Comput. Inf. Sci.* Poznań, Poland: Springer, 2018, pp. 141–149.

[8] W. Aoudi, M. Iturbe, and M. Almgren, "Truth will out: Departure-based process-level detection of stealthy attacks on control systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 817–831.

[9] Q. Cao, Y. Qiao, and Z. Lyu, "Machine learning to detect anomalies in Web log analysis," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 519–523.

[10] Y. Cui, Y. Sun, J. Hu, and G. Sheng, "A convolutional auto-encoder method for anomaly detection on system logs," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2018, pp. 3057–3062.

[11] A. Dreves, F. Facchinei, C. Kanzow, and S. Sagratella, "On the solution of the KKT conditions of generalized Nash equilibrium problems," *SIAM J. Optim.*, vol. 21, no. 3, pp. 1082–1108, Jul. 2011.

[12] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *Proc. IEEE 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 859–864.

[13] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.

[14] W. Fang, X. Tan, and D. Wilbur, "Application of intrusion detection technology in network safety based on machine learning," *Saf. Sci.*, vol. 124, Apr. 2020, Art. no. 104604.

[15] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *Proc. IEEE 26th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2015, pp. 24–34.

[16] C. Feng, V. R. Palleti, A. Mathur, and D. Chana, "A systematic framework to generate invariants for anomaly detection in industrial control systems," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Nov. 2019, pp. 1–22.

[17] J. Fontaine, C. Kappler, A. Shahid, and E. De Poorter, "Log-based intrusion detection for cloud Web applications using machine learning," in *Proc. Int. Conf. Parallel, Grid, Cloud Internet Comput.* Antwerp, Belgium: Springer, 2019, pp. 197–210.

[18] Q. Fu *et al.*, "Where do developers log? An empirical study on logging practices in industry," in *Proc. Companion Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 24–33.

[19] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "LogMine: Fast pattern recognition for log analytics," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2016, pp. 1573–1582.

[20] E. Hazan, "Introduction to online convex optimization," *Found. Trends Optim.*, vol. 2, nos. 3–4, pp. 157–325, 2016.

[21] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.

[22] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 60–70.

[23] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," 2020, *arXiv:2008.06448*. [Online]. Available: http://arxiv.org/abs/2008.06448

[24] S. Khatuya, N. Ganguly, J. Basak, M. Bharde, and B. Mitra, "ADELE: Anomaly detection from event log empiricism," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 2114–2122.

[25] T. Li *et al.*, "FLAP: An end-to-end event log analysis platform for system management," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1547–1556.

[26] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. 38th Int. Conf. Softw. Eng. Companion*, May 2016, pp. 102–111.

[27] Z. Liu, T. Qin, X. Guan, H. Jiang, and C. Wang, "An integrated method for anomaly detection from massive system logs," *IEEE Access*, vol. 6, pp. 30602–30611, 2018.

[28] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *Proc. USENIX Annu. Tech. Conf.*, 2010, pp. 1–14.

[29] J. Lu, F. Li, L. Li, and X. Feng, "CloudRaid: Hunting concurrency bugs in the cloud via log-mining," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Oct. 2018, pp. 3–14.

[30] H. Brendan McMahan, "A unified view of regularized dual averaging and mirror descent with implicit updates," 2010, *arXiv:1009.3240*. [Online]. Available: http://arxiv.org/abs/1009.3240

[31] W. Meng *et al.*, "LogParse: Making log parsing adaptive through word classification," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–9.

[32] W. Meng *et al.*, "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 4739–4745.

[33] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *Proc. 26th Conf. Program Comprehension*, May 2018, pp. 167–177.

[34] J. Mohan, A. Martinez, S. Ponnapalli, P. Raju, and V. Chidambaram, "Finding crash-consistency bugs with bounded black-box crash testing," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement.*, 2018, pp. 33–50.

[35] I. J. Myung, "Tutorial on maximum likelihood estimation," *J. Math. Psychol.*, vol. 47, pp. 90–100, Feb. 1985.

[36] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement.*, 2012, p. 26.

[37] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 215–224.

[38] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," 2020, *arXiv:2008.09340*. [Online]. Available: http://arxiv.org/abs/2008.09340

[39] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2007, pp. 575–584.

[40] R. Pokhrel, P. Pokharel, and A. Kumar Timalsina, "Anomaly-based–intrusion detection system using user profile generated from system logs," *Int. J. Sci. Res.*, vol. 9, no. 2, p. 8631, Feb. 2019.

[41] R. Vaarandi, B. Blumbergs, and M. Kont, "An unsupervised framework for detecting anomalous messages from syslog log files," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2018, pp. 1–6.
[42] J. Wang *et al.*, "An anomaly prediction framework for financial it systems using hybrid machine learning methods," *J. Ambient Intell. Hum. Comput.*, vol. 6, pp. 1–10, Dec. 2019.
[43] Q. Wang, J. Xu, H. Chen, and B. He, "Two improved continuous bag-of-word models," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2851–2856.
[44] Y. Wang, D. Li, Y. Du, and Z. Pan, "Anomaly detection in traffic using L1-norm minimization extreme learning machine," *Neurocomputing*, vol. 149, pp. 415–425, Feb. 2015.
[45] F. Wu, P. Anchuri, and Z. Li, "Structural event detection from log messages," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1175–1184.
[46] X. Xie, Z. Jin, Q. Han, S. Huang, and T. Li, "A confidence-guided anomaly detection approach jointly using multiple machine learning algorithms," in *Proc. Int. Symp. Cyberspace Saf. Secur.* Guangzhou, China: Springer, 2019, pp. 93–100.
[47] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. 22nd Symp. Oper. Syst. Princ.*, 2009, pp. 117–132.
[48] Y. Yuan, W. Shi, B. Liang, and B. Qin, "An approach to cloud execution failure diagnosis based on exception logs in OpenStack," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2019, pp. 124–131.
[49] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf.*, Aug. 2019, pp. 807–817.
[50] J. Zhu *et al.*, "Tools and benchmarks for automated log parsing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, May 2019, pp. 121–130.

Shangbin Han received the B.S. and M.S. degrees in electronic information science and technology from Shandong Normal University. He is currently pursuing the Ph.D. degree with Beihang University. His research interests include network security and AI.

Qianhong Wu received the Ph.D. degree in cryptography from Xidian University, in 2004. Since then, he has been with Wollongong University, Wollongong, NSW, Australia, as an Associate Research Fellow, with Wuhan University, China, as an Associate Professor, and with Universitat Rovira i Virgili, Spain, as a Research Director. He is currently a Professor with Beihang University, China. His research interests include cryptography, information security and privacy, VANET security, and cloud computing security. He has been a Holder/Co-Holder of China/Australia/Spain funded projects. He has authored 100 patents and more than 150 publications. He has served as an Associate/Guest Editor in several international ISI journals and in the program committee of dozens of international conferences.

Han Zhang (Member, IEEE) received the B.S. degree in computer science and technology from Jilin University and the Ph.D. degree from Tsinghua University. He is currently working with the School of Cyber Science and Technology, Beihang University. His research interests include computer networks, network security, and AI. He has published more than 30 articles in his area.

Bo Qin received the Ph.D. degree in cryptography from Xidian University, China, in 2008. Since then, she has been with the Xi'an University of Technology, China, as a Lecturer and with Universitat Rovira i Virgili, Catalonia, as a Post-Doctoral Researcher. She is currently an Associate Professor with Renmin University, China. Her research interests include pairing-based cryptography, data security and privacy, and VANET security. She has been a Holder/Co-Holder of seven China/Spain funded projects. She has authored more than 80 publications and served in the program committee of several international conferences in information security.

Jiankun Hu (Senior Member, IEEE) received the Ph.D. degree in control engineering from the Harbin Institute of Technology, China, in 1993, and the master's degree in computer science and software engineering from Monash University, Melbourne, VIC, Australia, in 2000. He was a Research Fellow with the Delft University of Technology, The Netherlands, from 1997 to 1998, and Melbourne University, Parkville, VIC, Australia, from 1998 to 1999. He has been with Ruhr University, Bochum, Germany. He is currently a Full Professor and the Research Director of the Cyber Security Laboratory, School of Engineering and Information Technology, University of New South Wales, Kensington, NSW, Australia. His main research interests include in the field of cyber security, including biometrics security, where he has published many papers in high-quality conferences and journals, including the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. He received the prestigious German Alexander von Humboldt Fellowship from Ruhr University from 1995 to 1996. He has received seven Australian Research Council (ARC) Grants, and serves at the prestigious Panel of Mathematics, Information, and Computing Sciences and the ARC Excellence in Research for Australia Evaluation Committee. He has served on the Editorial Board of up to seven international journals, including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and served as the Security Symposium Chair of the IEEE ICC and the IEEE GLOBECOM.

Xingang Shi (Member, IEEE) received the B.S. degree from Tsinghua University and the Ph.D. degree from The Chinese University of Hong Kong. He is currently working with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network measurement and routing protocols.

Linfeng Liu received the B.S. and Ph.D. degrees in computer science from the Southeast University, Nanjing, China, in 2003 and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, China. His main research interests include in the areas of multi-hop mobile wireless networks and vehicular ad hoc networks. He has published more than 60 peer-reviewed papers in some technical journals or conference proceedings, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (TPDS), *ACM Transactions on Autonomous and Adaptive Systems* (TAAS), IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY (TVT), *Computer Networks*, *Journal of Parallel and Distributed Computing*, *Journal of Network and Computer Applications*, IEEE SECON, GLOBECOM, ICC, and WCNC.

Xia Yin received the B.E., M.E., and Ph.D. degrees in computer science from Tsinghua University, in 1995, 1997, and 2000, respectively. She is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. Her research interests include future Internet architecture, formal method, protocol testing, and large-scale Internet routing.