Edge Intelligence (EI)-Enabled HTTP Anomaly Detection Framework for the Internet of Things (IoT)

Yufei An, F. Richard Yu, *Fellow, IEEE*, Jianqiang Li, Jianyong Chen, and Victor C. M. Leung, *Fellow, IEEE*

***Abstract*—In recent years, with the rapid development of the Internet of Things (IoT), various applications based on IoT have become more and more popular in industrial and living sectors. However, the hypertext transfer protocol (HTTP) as a popular application protocol used in various IoT applications faces a variety of security vulnerabilities. This article proposes a novel HTTP anomaly detection framework based on edge intelligence (EI) for IoT. In this framework, both clustering and classification methods are used to quickly and accurately detect anomalies in the HTTP traffic for IoT. Unlike the existing works relying on a centralized server to perform anomaly detection, with the recent advances in EI, the proposed framework distributes the entire detection process to different nodes. Moreover, a data processing method is proposed to divide the detection fields of HTTP data, which can eliminate redundant data and extract features from the fields of an HTTP header. Simulation results show that the proposed framework can significantly improve the speed and accuracy of HTTP anomaly detection, especially for unknown anomalies.**

***Index Terms*—Anomaly detection, edge intelligence (EI), hypertext transfer protocol (HTTP), Internet of Things (IoT).**

I. Introduction

INCREDIBLE developments in the routine use of network services and electronic applications have led to massive advances in communications networks and the emergence of the concept of the Internet of Things (IoT). IoT is a promising paradigm consisting of distributed sensor nodes, cloud servers, and software. Devices connected to IoT can be treated as objects or things for real-time sensing and processing, and they have the ability to sense their environments, connect with each other, and exchange data over the Internet. Such a network brings great economic benefits as it improves the efficiency of data generation and use. In recent years, IoT has made significant contributions to many areas, such as smart homes, healthcare, agriculture, transportation, and so on [1]–[3]. The number of IoT devices is projected to reach 20 billion by 2020, compared to the world population of over 7 billion.

The increasing number of IoT devices will provide many opportunities for attackers to compromise them through malicious emails, collusion attacks, and denial of service attacks, among many other types of attacks [4]. Many IoT devices are subject to various network attacks that exploit various vulnerabilities, such as encryption and password security. Intruders can control smart objects through cyber attacks and affect entire IoT networks by spreading malicious applications (such as malware). Thus, cyber security is now emerging as a key barrier to the more widespread adoption of IoT services.

There are generally two major categories of methods for detecting abnormal traffic in IoT: 1) signature and 2) anomaly-based detection [4]–[7]. Each category has its own benefits and limitations. Signature-based detection methods have very promising detection performance for known anomalies. However, they are challenged by new kinds of attacks, as the features of these attacks are hard to be identified. Anomaly-based detection builds a model containing samples of normal behaviors and considers deviation from the model to identify suspicious behaviors or attacks. These approaches can detect new types of attacks, but it may lead to an increase of false positives, mainly due to the uneven distribution of training samples.

Although some excellent works have been done on the security issues of IoT, most existing works focus on the security issues at the networking layers, such as authentication, encryption, key management, data consistency, etc. Consequently, *application* layer security has been largely ignored in the existing works. Particularly, the hypertext transfer protocol (HTTP) as a universal protocol has been widely used in IoT applications. However, due to the openness and diversity of IoT application protocols, it is easy to involve security vulnerabilities in the stages of design and deployment. HTTP can be exploited by attackers, which will seriously threaten users' personal information privacy and property in IoT. For example,

Manuscript received April 24, 2020; revised July 29, 2020; accepted September 2, 2020. Date of publication September 18, 2020; date of current version February 19, 2021. This work was supported in part by the National Nature Science Foundation of China under Grant U1713212, Grant 61836005, and Grant 61702341; in part by the Natural Science Foundation of Guangdong Province-Outstanding Youth Program under Grant 2019B151502018; in part by the Technology Research Project of Shenzhen City under Grant JSGG20180507182904693; and in part by the Public Technology Platform of Shenzhen City under Grant GGFW2018021118145859. *(Corresponding authors: Jianqiang Li; Victor C. M. Leung.)*

Yufei An, Jianqiang Li, and Jianyong Chen are with the Guangdong Laboratory of Artificial Intelligence and Digital Economy, Shenzhen University, Shenzhen 518060, China (e-mail: lijq@szu.edu.cn).

F. Richard Yu is the with the School of Information Technology, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: richard.yu@carleton.ca).

Victor C. M. Leung is with the Guangdong Laboratory of Artificial Intelligence and Digital Economy, Shenzhen University, Shenzhen 518060, China, and also with the Department of ECE, University of British Columbia, Vancouver, BC V6T1Z4, Canada (e-mail: vleung@ece.ubc.ca).

Digital Object Identifier 10.1109/JIOT.2020.3024645

malicious query codes can be embedded into URLs to launch HTTP attacks and obtain access to permission information [8].

In this article, with the recent advances in edge intelligence (EI) [9], [10], we study HTTP anomaly detection for IoT. The contributions of this article are summarized as follows.

1) A novel HTTP anomaly detection framework is designed to sequentially use both clustering and classification methods, which can quickly and accurately detect anomalies in the HTTP traffic for IoT.
2) Unlike the existing works relying on a centralized server to perform anomaly detection, with recent advances in EI and considering resource utilization efficiency, the proposed framework distributes the entire detection process to different nodes. This framework can efficiently relieve network congestion and computing pressures of centralized servers, as well as releasing the potential of EI in IoT.
3) A novel data processing method is proposed to divide the detection fields of HTTP data, which can eliminate redundant data and extract features from the fields of an HTTP header. In addition, a one-class HYBIRD classifier is presented to enhance the performance of the anomaly detection framework.
4) The simulation results are presented to show that the proposed framework can significantly improve the speed and accuracy of HTTP anomaly detection, especially for unknown anomalies.

The remainder of this article is structured as follows. Some related works of anomaly detection are briefly reviewed in Section II. Section III gives some preliminaries for the proposed framework while Section IV introduces the proposed framework in detail. The simulation results are presented in Section V. Finally, conclusions are presented in Section VI with future work.

II. Related Works

In this section, some related works about anomaly detection for HTTP traffic and EI are introduced.

A. Detection Methods

In the early studies, feature matching methods are mostly used to detect abnormal HTTP traffic. This kind of methods is simple and efficient but can only detect anomalies with known attacking features [6]. In order to detect some unknown anomalies, most of the related works use statistical methods or machining learning methods for anomaly detection, which are, respectively, introduced as follows.

1) Statistical Methods: Statistical methods detect anomalies mainly dependent on the predefined threshold, mean and standard deviation, and probabilities [11]. In [12], feature analysis is performed on the URL sample parameters to extract six characteristics of the request URL from the HTTP traffic, which is used to detect unknown behaviors. However, this approach can only detect a little of unknown anomalies. In [13], each Web session is divided into some subsessions with fixed length, on which the Bayesian estimation method is run to detect Web anomaly. In [14], a generalized-likelihood ratio test is designed to find out anomalies in network traffic, which exploits a nonrestricted α-stable first-order model and statistical hypothesis testing. In [15], data processing is first run on the HTTP traffic by using natural language processing and dimensionality reduction, and then a Gaussian model trained by normal data is adopted for anomaly detection. Although these approaches can achieve promising performance on detecting unknown anomalies, they have some evident shortcomings, such as high false positive rate and heavy computational cost.

2) Machine Learning Methods: Alongside the development of machine learning, many researchers use machine learning algorithms for anomaly based methods. Machine learning algorithms can be categorized into the following main categories, i.e., supervised learning, semisupervised learning, unsupervised learning, and reinforcement learning. It can learn the general features of the training traffic data. Based on the learned features, the input traffic data will be detected correctly. This generalized learning mode allows machine learning algorithms to process data that have never appeared before [16].

For supervised learning, it can be divided into classification problems and regression problems. People usually use classification algorithms to predict the category to which the data belong, and it needs labeled data to train the classifier. The most typical algorithms are the K-nearest neighbor (KNN) algorithm and the support vector machine (SVM) algorithm. In [17], a natural language processing method is used to extract features from the HTTP traffic, and then the SVM classification algorithm is adopted to train the detection model, which aims to detect malicious applications in Android devices. In [18], an artificial neural network and SVM classifier are combined through the classical data fusion method based on the conditional probability, which can reduce the error rate.

For semisupervised learning, it uses both a large amount of unlabeled data and a small amount of labeled data for pattern recognition. In [19], a detection model is constructed by using the modified Mahalanobis distance based on the principal component analysis (PCA), which is effective to detect anomalies in traffic. Similarly, a semisupervised classification method is presented in [20] by using density models based on the deep generative models and variational inference theory.

For unsupervised learning, clustering algorithms are extensively used for anomaly detection, which does not need to know the classification labels or the grouping of data categories. In [21], a clustering detection method based on seed extension is proposed to classify malicious traffic into different attacking phases so that anomalies can be identified in different environments. It first preprocesses the networks' traffic, including constructing the network flow, changing continuous-valued attributes into nominal attributes by adopting the discretization method, and further turning into binary features. Then, based on these features, it computes a weight for each flow and iteratively selects seeds to expand until all flows are divided into clusters. In [22], a graph clustering algorithm is used to detect attacks in network traffic. The numbers and weights of clusters are, respectively, calculated for the normal and malicious

network traffic graphs, and then their differences can be used to detect attacks. In [23], Android malware can be detected by clustering the HTTP traffic, which extracts features from the HTTP traffic and then adopts a fuzzy *C*-means clustering algorithm to give a good detection performance.

For reinforcement learning, it is through interaction with the environment to continuously improve behavior to maximize the concept of cumulative rewards. In [24], a novel reinforcement learning-based intrusion detection scheme was proposed to detect network traffic, and the detection model would become more accurate over time. Reinforcement learning for adaptive network defense is proposed in [25], which can avoid a certain pattern of attack and identify the pattern of attack from the cyber outlaw.

B. Edge Intelligence

With the popularity of IoT, a large number of devices are connected to the Internet, generating massive data at the edge of the network. In order to solve the computing and service pressures of the cloud data center and fully release the potential of big data at the edge, EI is beginning to be widely used. EI aims at coordinating a multitude of collaborative edge devices and serves to process the generated data in proximity [8]. It pushes computing tasks and services from the core of the network to the edge of the network, alleviating the pressure on the core of the network, and overcoming the problems of high monetary costs and high transmission delays that may occur when moving large amounts of data over a wide area network (WAN). Compared to the data center, edge servers are in closer proximity to people, data source, and servers. EI can bring a lot of benefits, including low latency, energy efficiency, privacy protection, reduced bandwidth consumption, etc. It has been applied in intrusion detection of IoT networks. In [26], a method for intrusion detection that combines distributed agents on Industrial IoT (IIoT) edge devices with centralized logging is proposed, which can improve the detection performance of the detection system. In [9], a new anomaly detection architecture for IIoT based on EI is proposed. It uses a new machine learning algorithm called LightGBM and traditional deep learning algorithms for detection.

C. One-Class Classification

In fact, in some scenarios, such as network intrusion detection, fault diagnosis, and fraud detection, normal data are far larger than anomaly data. As revealed in [7], the imbalance in training data has a significant impact on the results of anomaly detection. In this case, the one-class classification algorithm can be used to only identify one particular class by primarily learning from a training set containing only the objects of this class. In [27], a one-class SVM (OC-SVM) algorithm is proposed as the single classification problem. It is extended to detect the application layer DDoS attack in [28], which first extracts seven features from sessions of normal users and then models the browsing behavior of normal users by OC-SVM. A fusion of multiple one-class classifiers is presented in [29] to detect Web-based attacks. This approach uses a novel binary artificial bee colony algorithm to prune the initial ensemble of OC-SVM classifiers, which can find a near-optimal subensemble. Moreover, an improved OC-SVM method is designed in [30] with privileged information as the training data for malware detection, which performs better than the model using the training data without privileged information. In [31], a support vector data description (SVDD) classification algorithm [32] is also extended for detecting network anomalies by modeling the normal data.

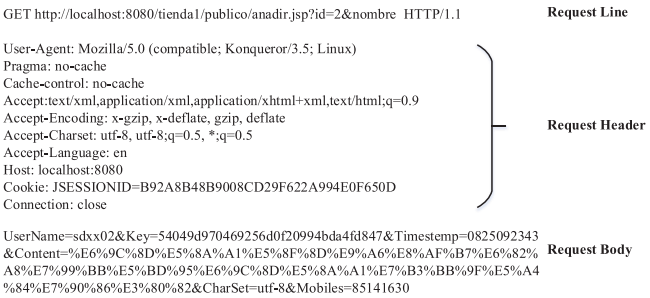

Fig. 1. Example of HTTP data.

III. PRELIMINARIES

Inspired by the one-class classification methods and most clustering algorithms used for anomaly detection, this article proposes a novel anomaly detection framework with sequential clustering and classification, which can well solve the data uneven problem when detecting malicious behaviors in IoT applications from the HTTP traffic. At the same time, we exploit the advantages of EI to design this detection framework, which is beneficial to relieve network congestion and computing pressures of centralized servers. As supported by the simulations in Section V, the proposed framework is very effective to improve the detection accuracy of unknown anomalies while keeping the low false positive rate. Before the presentation of our framework, the format of input data, i.e., the HTTP traffic, is first described. Then, we, respectively, introduce the clustering algorithm and the classification algorithm used in this article.

A. Data Description

The scenario studied in this article is malicious traffic detection in IoT applications. These applications request access to the Internet or transmit information via the HTTP protocol. Therefore, the format of the traffic data is the format of the HTTP data. In this section, the format of HTTP traffic is introduced in detail and some common attacks on the HTTP traffic are presented.

The HTTP request includes three components, i.e., request line, request header, and request body. The request line begins with a method token, followed by the request-URI and the protocol version, and ends with CRLF. The request header encoded by ASCII characters is comprised of structured fields. The request body can be encoded by different encoding types, which are determined by the specific content it carries [33]. The format of the HTTP data is illustrated in Fig. 1.

The request header contains different fields, such as accept-language and referer, in which intrusion attacks can be

TABLE I
INTRUSION ATTACK EXAMPLES

Filed Name	Attack Type	Example
User-Agent	Directory traversal	/./../../../../etc/passwd
X-Forwarded-For	SQL injections	-1' OR 3+534-534-1=0+0+ 0+1 or 'VWiuFOe3'='
Referer	XSS attack	http://dzx.wru.com.cn/x sstest;!@$[]"'~`^&*()

inserted. Several examples of intrusion attacks are listed in Table I. In this table, three typical attacks are introduced that occur in different fields of the request header. The first one is the directory traversal attack that occurs in the user-agent field. It utilizes insufficient security validation of users' input file names, which contain "/." In this case, malicious users can achieve directory jump, which can traverse to the parent directory on the server. This attack can grant an application the access to the files that are not allowed by the default privilege. In general, as a common file in Unix, the path "/etc/passwd" contains the password to demonstrate directory traversal. In order to detect this vulnerability, the URL that contains substrings like "../" or its escape character "x5c./" has to be restructured [34]. The second attack, i.e., SQL injection, can extract customer and order information from ecommerce databases or can bypass security mechanisms. The attacks simply inject operations that always result in true or false statements, by altering the predefined logical expressions within a predefined query. Typically, this injection can work through a Web form and an associated CGI script without appropriate input validation. These injections are not restricted to character fields. Similar alterations to the "where" and "having" SQL clauses have been exposed if the application does not limit numeric data for numeric fields [35]. The last one is the XSS attack as a very common application layer hacking technique, which will occur in the referer field. It allows an attacker to embed malicious JavaScript, VBScript, ActiveX, HTML, or Flash into a vulnerable dynamic page, which can cheat the user and execute the script on his/her machine in order to gather data [36]. The most common approach of stealing cookies or hijacking session is to embed a JavaScript encoded with the browser supported HTML encoding technique.

B. Birch Clustering Algorithm

In this article, we used a hierarchical clustering method called the Birch algorithm, which does not need to set any initial class or initial centroid. Based on the cluster feature tree (CF tree), it has high speed for clustering, especially for very large data sets. This advantage can fully meet the timing requirement of clustering the HTTP traffic data. The Birch clustering algorithm is briefly introduced as follows.

$\{\overrightarrow{X_i}\}$ represents N-dimensional data points in a cluster, where $i = 1, 2, \ldots, N$. Then, the clustering feature (CF) entry of the cluster is defined as a triple: $\mathrm{CF} = (N, \overrightarrow{LS}, SS)$, where N is the number of data points in a cluster, $\overrightarrow{LS}$ is the linear sum of N data points, i.e., $\sum_{i=1}^{n} \overrightarrow{X_i}$, and SS is the square sum of N data points, i.e., $\sum_{i=1}^{n} \overrightarrow{X_i}^2$ [37]. Assume that two disjoint subclusters have the entries $\mathrm{CF}_1 = (N_1, \overrightarrow{LS_1}, SS_1)$ and $\mathrm{CF}_2 = (N_2, \overrightarrow{LS_2}, SS_2)$. Then, the two disjoint subclusters are merged and the CF entry of this new subcluster is

$$\mathrm{CF}_1 + \mathrm{CF}_2 = \left(N_1 + N_2, \overrightarrow{LS_1} + \overrightarrow{LS_2}, SS_1 + SS_2\right). \quad (1)$$

Algorithm 1 Birch Clustering

Begin
  While $i > n$ $(i = 1)$ **do**
    Select the nearest child node as a data point from the root to the bottom
    After reaching the leaf node
    If the nearest tuple CF_i can absorb the data point then
      Update CF_i value
    Else
      If a new tuple can be added **then**
        Add a new tuple
      Else
        Split the farthest pair of tuples and take them as new leaf nodes. Then redistribute other tuples by the distance from the new leaf nodes
  Update tuple information of each non-leaf node
  If the node is split then
    Insert a new tuple in the parent node
End

The workflow of the Birch clustering algorithm is plotted in Algorithm 1. At the beginning, an initial in-memory CF-tree using the given amount of memory is built after all data are scanned. Starting from the root and recursively descending the CF-tree, choose the closest child node according to the distance D_0 as defined in

$$D_0 = \left| X_i - \frac{\sum_{i=1}^{n} \overrightarrow{X_i}}{N} \right|. \quad (2)$$

Upon reaching a leaf node, find the closest leaf entry, say X_1, and then examine whether X_1 can "absorb" X_2 without violating the threshold condition. If so, the CF entry for X_1 is updated to reflect this. If not, a new entry for X_2 is added to the leaf. If this new entry has space on the leaf to fit in, the insert operation is done. Otherwise, the leaf node must be split. The node is split by choosing the farthest pair of entries as seeds and then redistributing the remaining entries based on the closest criteria. After X_2 is inserted into a leaf, the CF information is updated for each nonleaf entry on the path to the leaf. If a split does not occur, it simply involves updating the existing CF entries to reflect the addition of X_2. To describe the newly created leaf, a leaf split requires inserting a new nonleaf entry into the parent node. If the parent has space for this entry, at all higher levels, the CF entries are updated to reflect the addition of X_2. On up to the root, if the root is split, the height of CF-tree is increased by one [37].

C. OC-SVM

OC-SVM is developed based on the traditional SVM. It is used to solve a specific problem with only one class of samples

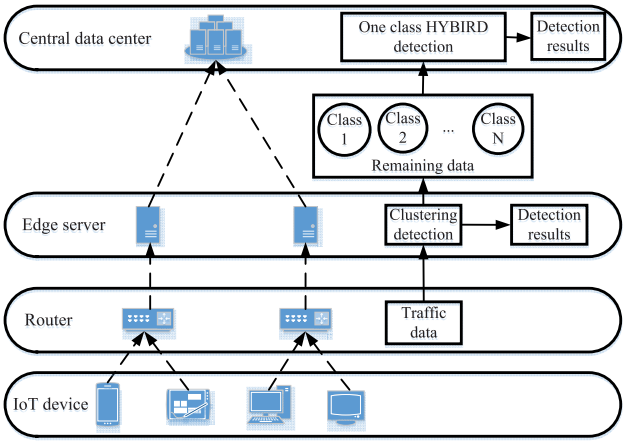

Fig. 2. Proposed HTTP anomaly detection framework.

to train the classification model [27]. Given a training data set $\{X_i\}_{i=1}^n$ with $X_i \in R^n$, there is a nonlinear mapping Φ from R^n to a high-dimensional feature space χ such that $\Phi(X_i) \in \chi$. Then, a hyperplane is created in the high-dimensional space with $\omega \cdot \Phi(X) - \rho = 0$, where ω is the normal vector of the hyperplane and ρ is the intercept of the hyperplane. It indicates that the mapped samples are separated from the origin by the interval ρ. In order to keep the hyperplane away from the origin, the Euclidean distance ρ/ω between the origin and the target data is used to find the optimal hyperplane. Then, to separate the data set from the origin, the following quadratic programming problem needs to be solved:

$$\textbf{min} \quad \frac{1}{2}\omega^2 + \frac{1}{\upsilon n}\sum_{i=1}^{n}\xi_i - \rho \tag{3}$$

$$\text{subject to} \quad (\omega \cdot \Phi(X_i)) \geq \rho - \xi_i, \quad i = 1, 2, \ldots, l, \quad \xi_i \geq 0 \tag{4}$$

where n is the number of data points, ξ_i is the nonnegative slack variable of X_i, and $\upsilon \in (0, 1]$ is a regularization parameter to control the fraction of outliers. ω and ρ are the parameters that determine the decision boundary. With the Lagrange function [27], (3) can be transformed into

$$f(x) = \text{sign}((\omega \cdot \Phi(X) - \rho)). \tag{5}$$

IV. HTTP Anomaly Detection Framework

In this article, a novel anomaly detection framework is proposed based on the above Birch clustering and one-class classification algorithms, which can quickly and accurately detect anomalies from the HTTP traffic. Here, the overall detection framework is provided in Section IV-A. Then, three main parts in our framework, i.e., data processing, clustering process, and classification process, are, respectively, introduced in Sections IV-B–IV-D.

A. Proposed Detection Framework

An overview of the proposed detection framework is shown in Fig. 2. The framework can be divided into two parts: 1) the cluster detection process and 2) the classification detection process. The details of each process are further provided in Fig. 3. Traffic data from IoT devices will go through the first step of the cluster detection process. Before this process, traffic data and training data need to complete data preprocessing tasks in the data processing module. The specific description of the data processing module will be presented in Section IV-B. Then, we use the Birch cluster algorithm to train a cluster detection model to detect traffic data. After this step of detecting, some normal data will be successfully detected, and the remaining data will be divided into N classes, which will be detected during the classification detection process. During the classification detection process, we employ a new machine learning algorithm, one-class HYBIRD, for intrusion detection of remaining data. This algorithm consists of OC-SVM and the improved SVDD algorithm, which can improve the performance of the detection framework. In this process, we first extract features from each class and then use normal data detected by the cluster detection process to train the classifier. Eventually, each class in the remaining data will be detected by the classifier. Through these two processes, all data will be detected.

The IoT network can be divided into a central network part and an edge part. In the IoT network, we regard a central data center with powerful computing capabilities and sufficient resources as the master node while the equipment of the edge part (i.e., edge serves) is regarded as an edge node. Due to limited computing power and resources of edge nodes, in order to improve energy efficiency, we perform two detection processes on different nodes, respectively. Cluster detection is performed on edge nodes, and classification detection is performed on master nodes.

When the traffic data of IoT applications are transmitted from the devices to edge nodes through the routers, the edge servers will perform some data preprocessing work, extract the relevant features of the traffic, and then apply a clustering algorithm to detect. The Birch clustering algorithm we used in this process has the advantages of fast clustering speed and small memory consumption and is very suitable for running on edge nodes. So this process does not need to consume too many computing resources and can be completed quickly. In the master node, it also needs to perform feature extraction and train a complex classifier. Nevertheless, since the clustering detection divides the remaining data into different classes, each class contains data with similar characteristics. Consequently, when the master node performs detection, it only needs to perform feature extraction and detection on a specific class. This greatly reduces the workload of the master node so that the master node has sufficient resources to meet the needs of the training complex classifiers. Moreover, cluster detection on edge nodes removes some redundant data from the traffic data, which helps improve the accuracy of classification detection on the master node. Therefore, the master node can complete detection tasks accurately and efficiently.

The proposed detection framework fully combines the advantages of EI to split the entire detection process into different nodes. Without increasing the detection time, it effectively relieves the computing pressure of the data center. In addition, cluster detection in the framework filters some normal traffic, which effectively reduces the bandwidth burden.

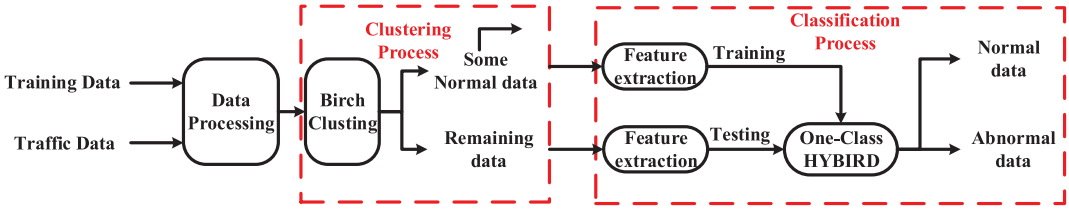

Fig. 3. Two detection processes.

```
GET http://zfxxgk.beijing.gov.cn/myq11SH00/jgsz12j/mybm_list.shtml HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko)
Cache-control: no-cache
Accept: */*
Accept-Encoding: gzip
Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
Accept-Language: zh-CN,en,*
Host: zfxxgk.beijing.gov.cn
X-Forwarded-For: 59.44.201.158
Referer: http://zfxxgk.beijing.gov.cn/myq11SH00/jgsz12j/mybm_list.shtml
Cookie: __jsluid=3d1fe437aaec9a5107593c15b1f7faa8

{"accept-language": "zh-CN,en,*",
 "accept": "*/*",
 "accept-encoding": "gzip",
 "accept-charset": "utf-8, utf-8;q=0.5, *;q=0.5",
 "cache-control": "no-cache",
 "x-forwarded-for": "59.44.201.158",
 "user-agent": "Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36
        (KHTML, like Gecko)
 "url": "/default/xhtml/myqzf/js/page.js",
 "host": "zfxxgk.beijing.gov.cn",
 "cookie": "__jsluid=3d1fe437aaec9a5107593c15b1f7faa8",
 "referer": "http://zfxxgk.beijing.gov.cn/myq11SH00/jgsz12j/mybm_list.shtml"}
```

Fig. 4. Example of different fields in HTTP data after segmentation.

B. Data Processing

HTTP data need to be processed before the detection, as shown in Fig. 3. It is mainly divided into three steps: 1) field detection; 2) normalization; and 3) vectorization. The details of each step are described as follows.

1) Field Detection: The HTTP traffic contains information from many different fields and each field may contain actual intrusion attacks. In order to better detect attacks and avoid interference caused by useless information in other fields, we propose field detection. Because each field of the header is detected separately, the header needs to be segmented. This method is good for eliminating redundant data and improving detection accuracy, and it can also identify the location of an anomaly in an exact field. Fig. 4 shows an example of different fields in HTTP data after segmentation.

Since the amount of HTTP data is very large, in order to ensure that data can be quickly accessed during the detection process, dictionary structure is used to represent the data. A dictionary named Dict_m is created. It has a series of keys and each key represents an HTTP header field. Each segment of the header is stored in Dict_m. In this way, in the process of field detection, a key representing a field can be quickly identified and the corresponding field can be obtained immediately. The complete HTTP data are shown in the upper area of Fig. 4. After segmentation, the data of each field are stored in the dictionary, which are shown in the lower area of Fig. 4. After all the data are segmented, each field is detected step by step.

TABLE II
NORMALIZATION EXAMPLES

Raw data	Form after normalization (Sig)
http://www.szrch.com:80	A://A.A.A:A
set—set&set	A—A&A
http://dzx.wru.com.cn/xss test;!@$[]"'~^&*()	A://A.A.A.A/A;!@$[]"'~^&*()

2) Normalization: In order to simplify the detection data and effectively protect the privacy of personal information, we have normalized the HTTP data. Because each field has some structural features and structural information, they can be simplified with some specific characters. Here, "C" is used to represent the SQL-related keywords in the data (e.g., "select," "count," "from," and "where"). All the remaining characters except symbols are replaced by "A." After normalization, the fields of an HTTP request header are transformed into sigs. It does not only simplify the data but also preserve its structural information. Some examples of normalization are shown in Table II, where the left-hand side is the fields' content to be normalized while the right-hand side is the corresponding normalized results.

3) Vectorization: In the vectorization phase, a sig is transformed into a vector to facilitate the subsequent clustering and classification stages. In this process, n-gram analysis is used to extract meaningful features from the sig. It could also be taken as a substring with the length n. For example, the string "ababc" contains four substrings with "ab," "ba," "ab," and "bc," which have three unique 2-grams "ab," "ba," and "bc." The 2-gram "ab" appears twice, with a frequency of 2/4. The 2-gram "ba" and "bc" only appears once, with a frequency of 1/4. A list of text tokens can be represented with a vector consisting of n-gram frequencies. A feature vector describing this string would be $X_{ababc} = [1/2, 1/4, 1/4]$. All sigs can be converted to vectors by this method. The normalized sig contains 36 different unique characters. After the n-gram analysis for these characters, we can get a bag-of-words, with which each sig can be transformed into a vector [17]. If $n = 1$ is chosen for the n-gram analysis, it is easy to lose the correlation among characters that may lead to poor performance of the detection framework. However, when n is larger than 2, the dimension of bag-of-words is increased dramatically. Since the performance of the detection framework strongly relates to the dimension of the feature vector, a high dimension of the feature vector would result in poor performance. Therefore, in this article, $n = 2$ is used, which means that the number of dimensions in

	A	.	(	.....	:	/
A	AA	A.	A(	.....	A:	A/
.	.A	..	.(	.....	.:	./
(	(A	(.	((	.....	(:	(/
.....	.....	.....	.....	.....	.....	.....
:	:A	:.	:(	.....	::	:/
/	/A	/.	/(	.....	/:	//

Fig. 5. Bag-of-words from 2-gram analysis.

TABLE III
VECTORIZATION EXAMPLE

Element	A :	: /	//	/A	A.	.A	others
The number of occurrences	1	1	1	1	4	4	0
Frequency	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{4}{12}$	$\frac{4}{12}$	0

the feature vector is $36^2 = 1296$. More simulation evaluations will be introduced in detail in Section V-G.

Fig. 5 shows the bag-of-words got from the 2-gram analysis. A vector is produced with the same length of the bag-of-words. Moreover, each element of the vector represents an occurrence frequency of the corresponding symbol in the bag-of-words with the same order. Then, a sig is also processed by the 2-gram analysis, and a 2-gram sequence can be obtained that represents the sig. The occurrence frequency of each symbol in the 2-gram sequence is calculated and becomes an element value of the vector whose location represents that symbol. If a symbol in the bag-of-words does not appear in the 2-gram sequence, the corresponding element in the vector is 0. With this method, each sig can be transformed into a numeric vector with a dimension that is equal to the length of the bag-of-words. Here, an example is given to show the transformation for a sig with "A://A.A.A.A.A." From this sig, a unique 2-grams sequence can be obtained with "A:," ":/," "//," "/A," "A.," and ".A" for a total of 12 substrings, and their occurrence frequency is shown in Table III. From this information and the bag-of-words shown in Fig. 5, a vector can be obtained with $v_1 = [0, 4/12, 0, \ldots, 0, 1/12, 0, 4/12, 0, \ldots, 0, 1/12, 1/12, 0, \ldots, 0, 1/12]$.

C. Clustering Process

In this process, the Birch clustering algorithm is used to divide the sigs converted from traffic data and training data into different classes according to their features, as shown in Fig. 3. For each class, if it contains the training sigs that are converted from training data, all sigs of this class belong to normal sigs. Otherwise, they belong to unknown sigs, which will be further classified in the next classification stage. An example is given in Fig. 6, where the red dots are training sigs and the blue dots are sigs, which are transformed from the HTTP traffic data. Each circle represents a class and all sigs are divided into 11 classes. There are five classes only containing the red dots, three classes only containing the blue dots, and three classes containing both the red and blue dots. For red circles that include the red dots, their sigs are taken as normal sigs. Thus, there are eight classes regarded as normal sigs in Fig. 6. For black circles that only contain the blue points, their sigs are taken as unknown sigs. The data corresponding to the normal sigs are taken as normal data while that corresponding to unknown sigs are taken as remaining data, which will be further detected in the next process.

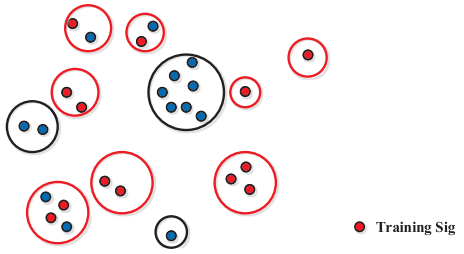

Fig. 6. Example of the clustering detection.

TABLE IV
NOTATIONS (ORDERED IN APPEARING SEQUENCE)

Notation	Definition
S	Set of sigs converted from data
T	The number of normal sigs misjudged as abnormal sigs
$dist_i$	Distance from train point to hyperplane
C_i	Penalty coefficient corresponding to train point
z	New sample point

D. Classification Process

In the second process of detection shown in Fig. 3, the normal data are used to train the proposed one-class HYBIRD classifier and the remaining data are further identified by the classifier. The stage is divided into two steps, i.e., feature extraction and the proposed one-class HYBIRD classification, which are, respectively, introduced below. Table IV summarizes the notations used in the following introduction.

1) Feature Extraction: In order to extract features from the corpus, i.e., both normal sigs and unknown sigs, the doc2vec algorithm [38] is used. It learns features from the corpus in an unsupervised manner and then provides a fixed-length feature vector as output. The corpus consists of a series of normal sigs and unknown sigs. For sigs $S_1, S_2, S_3, \ldots, S_n$ in a corpus, the following objective function is used to train the corpus:

$$\frac{1}{T}\sum_{i=k}^{T-k} \log p(S_t|S_{t-k}, \ldots, S_{t+k}) \tag{6}$$

where k is the window size for preserving the contextual information [38]. Then, using the softmax function, the vector of each sig can be calculated, as follows:

$$p(S_t|S_{t-k}, \ldots, S_{t+k}) = \frac{e^{y_{S_t}}}{\sum_i e^{y_i}} \tag{7}$$

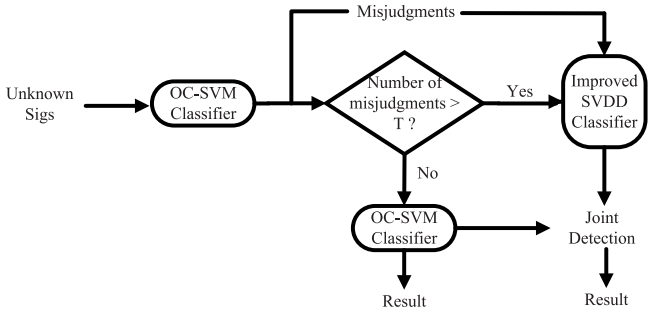

Fig. 7. Process of the one-class HYBIRD.

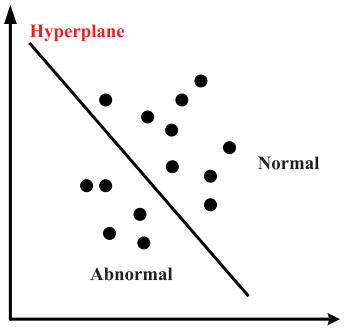

Fig. 8. Misjudged sigs by the OC-SVM classifier.

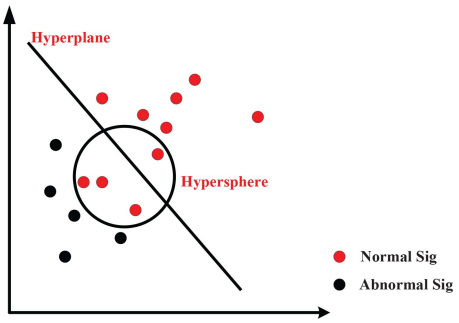

Fig. 9. Joint detection with OC-SVM and improved SVDD classifiers.

where each y_i is the ith output value of a feedforward neural network computed using [38], as follows:

$$y = b + Uh(S_t|S_{t-k}, \ldots, S_{t+k}; W, D) \tag{8}$$

where b and U are bias and weight matrix between the hidden and the output layers, and h is the average of sig vectors extracted from W, which is embedding matrix of sigs [38]. Each class is assigned a unique ID. D is the matrix of IDs representing sigs in the corpus, which come from different classes.

2) One-Class HYBIRD: In this article, we propose a new machine learning algorithm named one-class HYBIRD, which is composed of the OC-SVM algorithm and an improved SVDD algorithm. The detailed process of the one-class HYBIRD is shown in Fig. 7. First, we use the results from cluster detection to train the OC-SVM classifier. Then, some normal sigs will be used to verify the classification effect of this classifier. At this time, partial normal sigs may be misjudged as abnormal sigs. Fig. 8 shows a scenario of misjudgment of the normal sigs. All points in Fig. 8 are normal sigs, but the ones below the hyperplane are misjudged as abnormal sigs by the classifier. If the number of points below the hyperplane is very large, the performance of the OC-SVM classifier may be seriously degraded. Therefore, a threshold T is set and the number of misjudged sigs from the OC-SVM classifier is compared with T.

If the number of misjudged sigs is larger than threshold T, the misjudged sigs are used to train the improved SVDD classifier. After the training process, the unknown sigs are jointly detected by both the OC-SVM classifier and the improved SVDD classifier. The principle of a joint detection is shown in Fig. 9. In this figure, the red dot represents normal sig and the black dot represents abnormal sig. The hyperplane is the border of classification by using the OC-SVM classifier while hypersphere is the border of classification by using the improved SVDD classifier. The sigs above the hyperplane are all taken as normal sigs. For the sigs below the hyperplane, if they are inside the hypersphere, i.e., the dots inside the circle, they are also taken as normal sigs, otherwise, they are taken as abnormal sigs.

If the number of misjudged data is not larger than threshold T, the unknown sigs are only classified by the OC-SVM classifier. This is because the algorithm may not be able to train an effective classification model when the number of misjudged sigs is too small, which will affect the final detection result and increase the detection time.

3) Improved SVDD Algorithm: Given training data set $\{X_i\}_{i=1}^n$ with $X_i \in R^n$, the hypersphere of the improved SVDD algorithm can be obtained by solving the following optimization problems:

$$\textbf{min} \quad R^2 + C\sum_{i=1}^{l}\xi_i \tag{9}$$

$$\Phi(x_i) - a^2 \le R^2 + \xi_i, \quad \xi_i \ge 0; \quad i = 1, 2, \ldots, l \tag{10}$$

where parameter a is a linear combination of support vectors that represents the center of the hypersphere, R represents the radius, R^2 is the volume of hypersphere, and Φ is a function that maps data to the high-dimensional space. The distances of all data points x_i and the center should be less than R [32], and at the same time, a slack variable ξ_i is constructed with a penalty coefficient C. First, the distances (dist_i, $i = 1, \ldots, m$) are measured between all trained points and a hyperplane. The hyperplane is found by the OC-SVM classifier. The distance from each point to the hyperplane is taken as the penalty coefficient for that point. The distance size should be normalized and its value must be within the range of C (in this article, the range of C is between 0 and 1). The normalization method is done as follows:

$$C_i = \frac{\text{dist}_i - \text{dist}_{\min}}{\text{dist}_{\max} - \text{dist}_{\min}}. \tag{11}$$

Then, the Lagrange multiplier method is used to get the Lagrange function. Thus, the optimization problems can be

expressed as

$$L(R, a, \alpha, \xi, \gamma) = R^2 + \sum_{i=1}^{m} C_i \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \Big(R^2 + \xi_i \Big(x^{(i)^T} x^{(i)} - 2a^T x^{(i)} + a^2 \Big) \Big) - \sum_{i=1}^{m} \gamma_i \xi_i \tag{12}$$

where α_i and ξ_i are Lagrange multiplier. The Lagrange function has a partial derivative to zero for R, a, and ξ_i. Then, we have

$$\sum_{i=1}^{m} \alpha_i = 1$$
$$a = \sum_{i=1}^{m} \alpha_i x^{(i)}$$
$$\frac{\sum_{i=1}^{m} C_i}{m} - \alpha_i - \gamma_i = 0. \tag{13}$$

Finally, the problem to be solved becomes

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i x^{(i)^T} x^{(i)} - \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j x^{(i)^T} x^{(i)} \tag{14}$$

subject to

$$0 \le \alpha_i \le \frac{\sum_{i=1}^{m} C_i}{m}$$
$$\sum_{i=1}^{m} \alpha_i = 1, \quad i = 1, 2, \ldots, m. \tag{15}$$

For a new sample point z, if it satisfies the following formula, it can be taken as an abnormal point

$$(z - a)^T (z - a) > R^2. \tag{16}$$

Different from the traditional SVDD algorithm in which only the static C is used, the improved SVDD algorithm uses a dynamic C to replace the static one. In this article, different fields of the HTTP request header are classified by the improved SVDD algorithm. If the traditional SVDD algorithm is used, its optimal value of C may be different. For example, for the URL field, the algorithm achieves the best classification when $C = 0.5$ while for the Cookie field, it gets the best classification when $C = 0.8$. If a single value of C is used for all fields, the algorithm cannot guarantee the best overall performance. Thus, the improved SVDD algorithm is proposed with the dynamic C got from (11) directly in this article, which can get better performance than the traditional one.

V. Simulation Results and Discussions

We use Python and a popular machine learning library Scikit-learn to implement the proposed framework. In this section, we first introduce the used data sets. Next, the criteria for evaluation are introduced. Subsequently, we analyze the number of training samples for the detection framework to achieve good detection results. Then, our framework is compared with other methods to verify its feasibility. After that, simulations show that both the proposed data processed by field detection and one-class HYBIRD can improve the detection accuracy. Finally, in order to get good performance, the key parameters T in the framework are also analyzed. Performance and time results are based on an average of ten measurement runs.

TABLE V
Details of Data set

Items	Number
Normal Data	1,943,994
Malicious Data	314,420
Training Data	800,000
Test Data	1,458,414
Total Data	2,258,414

A. Data Sets

The HTTP data set we used in the simulations was collected from the deployed IoT devices. We used a variety of antivirus engines to obtain some normal data, and the remaining suspicious data were analyzed and filtered by the VirusTotal tool [39]. We extract the target IP field, domain name, or URL of each HTTP traffic and upload it to VirusTotal. If the domain name, target IP, or URL is malicious, the traffic containing these is considered abnormal traffic. The details of the data set are shown in Table V.

B. Performance Evaluation

The rates of precision, recall, and accuracy have been widely used to evaluate the performance of intrusion detection. Their definitions are given as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{17}$$
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{18}$$
$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}. \tag{19}$$

In the above equations, true positives (TP) are the number of normal events that are correctly classified as normal events.

True negatives (TN) are the number of anomalies that are correctly classified as anomalies.

False positives (FP) are the number of normal events that are wrongly classified as anomalies.

False negatives (FN) are the number of anomalies that are wrongly classified as normal events.

A low precision indicates that the intrusion detection system has a high false positive rate, which misjudges a large amount of normal data as abnormal and increases the workload of security managers. A low recall rate indicates that a large number of anomalies are not correctly detected, degrading the overall security performance of the system. Accuracy is the total number of normal event and anomalies correctly classified by the total number of data set. It is an indicator of the overall performance of the measurement model.

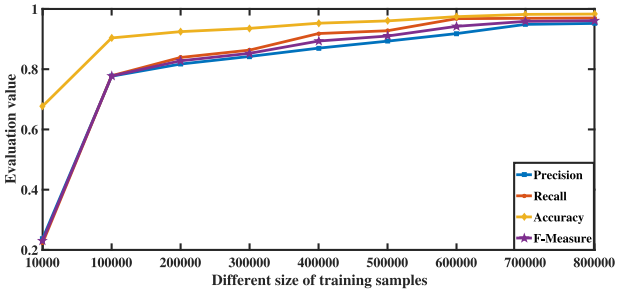

Fig. 10. Performance comparison of different training sample size.

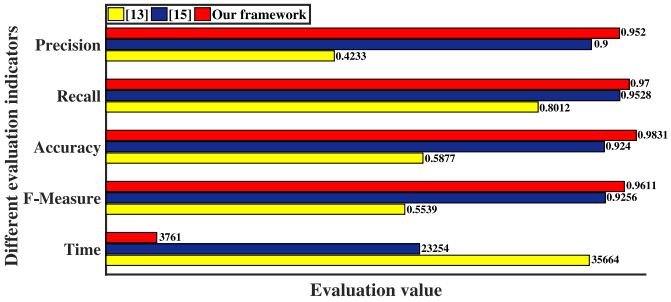

Fig. 11. Performance comparison of different methods.

However, the HTTP anomaly detection is actually an imbalanced classification problem, for imbalanced problems, precision, recall, and accuracy cannot accurately measure whether the model is good or bad, so we chose a relatively fair and better indicator F-measure to understand and compare the performance of malicious traffic detection when compared with accuracy. It involves the detection rate and error rate of identifying malicious samples, which is a combination of precision and recall, so it can better evaluate the detection performance. As shown in

$$F_\beta = \frac{(\beta^2+1)PR}{\beta^2 P + R} \tag{20}$$

where β is a parameter, P is precision, and R is recall. Usually, $\beta = 1$.

We, therefore, utilize the F-measure value as the most important metric in the following experimental evaluation. At the same time, using precision, recall, accuracy, and detection time as auxiliary evaluation parameters makes the experimental results more convincing.

C. Evaluation of Training Sample Size

In order to determine how many training samples are required for the detection framework to reliably detect malicious traffic, we designed a simulation experiment. Fig. 10 shows the results of the simulation. In Fig. 10, the abscissa represents the size of the training sample from 10 000 to 800 000. The ordinate corresponds to the indicators of precision, recall, accuracy, and F-measure for different training sample sizes. As the size of the training set increases, the four indicators continue to increase, but eventually the trend of increase is stable.

Specifically, when the training set contains only 10 000 samples, the detection effect is very poor. When the size of the training sample increases to 100 000, the detection effect of the model is greatly improved, the recognition rate of malicious traffic reaches 77.83% and the accuracy rate reaches 90.38%; When the training set contains 700 000 samples, the recognition rate of malicious samples reaches 96.9%, and the accuracy rate reaches 98.2%; when using all training samples (800 000 samples), the detection rate of the model for malicious traffic is 97%, and the accuracy is 98.31%. The experimental results indicate that an accurate and reliable detection framework model can be trained without many training samples.

D. Comparison With the Existing Approaches

To evaluate the performance of the proposed framework, we have chosen two methods that were reported recently for comparison, i.e., the methods from [15] and [17]. Juvonen *et al.* [15] proposed a method to detect HTTP traffic data by using PCA to reduce dimensionality and using the Gaussian model to train the detection model. Wang *et al.* [17] presented an intrusion detection method by using natural language processing and SVM. Their comparison results are shown in Fig. 11.

In Fig. 11, the vertical ordinate represents different evaluation indicators, i.e., precision, recall, accuracy, F-measure, and time (where the basic unit of time is seconds). The abscissa shows the values of these indicators. The yellow, blue, and red bars represent the methods in [15] and [17], and the proposed framework, respectively.

From Fig. 11, we can see that the proposed method has the highest values for the four indicators, and the detection time is the shortest, which validates that its performance is the best. Compared with the methods in [15] and [17], the proposed algorithm has three major different mechanisms for this performance improvement. First, the proposed anomaly detection framework sequentially runs the clustering algorithm and the classification algorithm in an alternative manner while the other two methods only use one single detection operation. Second, the proposed data processing with field detection is used to reduce redundant fields in HTTP request headers so that anomalies can be found easier while the other two methods just use all the HTTP traffic for detection, which makes them relatively difficult to find anomalies. Although the method in [17] also uses some technologies to reduce some redundant data, there are still a lot of redundancies in the HTTP traffic, which can evidently degrade the detection performance. Third, the proposed one-class HYBIRD classifier can reduce the false positive caused by the OC-SVM classifier, which further improves the detection accuracy.

Although our detection framework uses two machine learning methods, clustering and classification, our detection time is still the shortest. This is because the data processing with field detection used in our framework has removed some redundant fields before the anomaly detection. At the same time, during the detection process, all fields are normalized, which further simplifies the detection. Moreover, we have applied EI technology in the framework so that the process of cluster detection and classification detection for different fields can be

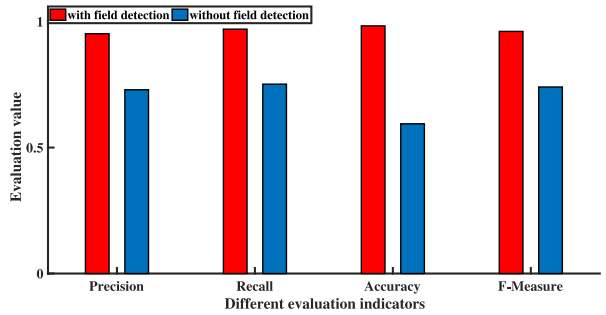

Fig. 12. Performance of the field detection method.

synchronized. These methods all help to shorten the detection time, so the detection time of our framework is the shortest.

E. Evaluation of Data Processing With Field Detection

In order to evaluate the function of data processing with field detection, experiments are run by comparing the proposed frameworks with and without data processing with field detection. Fig. 12 shows the simulation results, where the red bar indicates the method with the field detection method and the blue bar indicates the method without the field detection method. The F-measure indicator as well as the other three indicators show that the data processing with field detection can evidently improve the overall performance of the proposed framework. On the one hand, HTTP traffic always has a large amount of redundant data, which may affect the detection accuracy. With the method of field detection, most of the redundant data can be removed, which helps to improve the detection accuracy. On the other hand, since the anomalous characteristics identified in each field are more obvious than that identified in the entire HTTP traffic, it is easier for the data processing with field detection to find anomalies existing in a specific field. Therefore, data processing with field detection is able to improve the detection results from the above two aspects. Moreover, using field detection, it can also locate the field at which attacks occur, which is easier for the firewall to prevent attacks.

F. Evaluation of One-Class HYBIRD

In order to study the effectiveness of one-class HYBRID, we compare the detection results of the OC-SVM classifier and SVDD classifier. At the same time, in order to make the simulation results more convincing, in addition to conducting simulation under the data set we collected, we also introduced the CSIC data set to further evaluate the one-class HYBRID algorithm. This data set contains tens of thousands of automatically generated Web requests produced by the Spanish Research Council Information Security Institute [40]. However, because the data set is made by hand, it does not meet the actual situation. Therefore, we only use this data set to evaluate the performance of the classifier, but not to evaluate the performance of the detection framework. Table VI shows the comparison results on the CSIC and our data sets.

For two data sets, the one-class HYBIRD classifier is better regarding the precision, accuracy, and F-measure indicators.

TABLE VI
PERFORMANCE COMPARISON OF DIFFERENT CLASSIFIERS IN DIFFERENT DATA SETS

Dataset	Method	Precision	Recall	Accuracy	F-measure
Our dataset	One-class HYBRID	0.952	0.97	0.9831	0.9611
	Only OC-SVM	0.8944	0.9867	0.972	0.9382
	Only SVDD	0.7472	0.9948	0.9263	0.8534
CSIC	One-class HYBRID	0.9711	0.724	0.8779	0.8296
	Only OC-SVM	0.9168	0.7335	0.8633	0.815
	Only SVDD	0.9264	0.4663	0.7658	0.6204

In terms of precision, it means that the one-class HYBIRD classifier has a much lower false positive rate than the other classifiers. Although the recall value obtained by the one-class HYBIRD classifier is slightly lower than other classifiers, it does not affect the better performance of the one-class HYBIRD, as its higher F-measure value can reflect the better overall performance, which is more important than the recall indicator.

Due to the problem of false positives caused by the OC-SVM classifier, some normal data may be misjudged as abnormal ones, which will lead to lower detection accuracy. After using the one-class HYBIRD classifier, we can further detect the misjudged data to improve the detection accuracy. For a classifier, if the precision indicator is improved, the recall indicator may be affected. However, for the overall performance of the classifier, the improvement on the precision indicator is more important and more meaningful than the impact on the recall indicator. Therefore, the F-measure value of the one-class HYBIRD classifier is larger than that of the OC-SVM classifier. In other words, when compared to the OC-SVM classifier, the proposed one-class HYBIRD classifier is better to improve the overall detection performance. Similarly, the one-class HYBIRD classifier also has better performance than using only the SVDD classifier.

In the case of uneven sample distribution, a one-class classification algorithm usually brings better detection results. In the detection of IoT anomalies, normal HTTP traffic is often easier to obtain than abnormal traffic, so the use of one-class classification detection also greatly reduces the difficulty of collecting training data.

G. Different N-Gram Methods

The final detection result of the detection framework is affected by different N-gram methods. Table VII shows the F-measure values, recall, and detection times for different N values. Recall represents the detection rate, and detection time represents the computation cost. The longer the time,

TABLE VII
PERFORMANCE COMPARISON OF DIFFERENT N VALUES IN N-GRAM ANALYSIS

N-gram	Number of features	F-measure	Recall	Time (s)
1-gram	36	0.044	0.023	2003
2-gram	1296	0.9611	0.97	3761
3-gram	46656	0.9665	0.952	51647

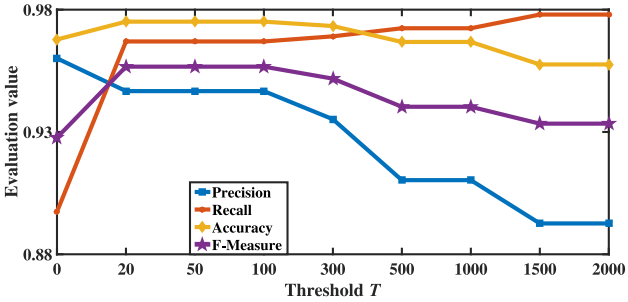

Fig. 13. Simulation results of different thresholds of T.

the higher the computation cost. As shown in the table, when $N = 1$, the number of features is only 36, and the detection time is the shortest, but the detection effect is very poor. The detection rate is only 2.3%, and the F-measure value is only 0.044. When $N = 2$, the number of features increases to 1296, the detection rate increases to 97%, the F-measure value increases to 0.9611, and the detection time does not rise much, which is 3761. When $N = 3$, the number of features becomes very large, which is 46 656. The detection rate is reduced to 95.2%, and the F-measure value is slightly increased to 0.9665. However, the price of a larger F-measure value is the increase in computation cost. At this time, the detection time is 51 647, which is an increase of almost 14 times compared with that when $N = 2$. When N becomes larger, the computation cost will increase more, and the detection rate will also decrease. Therefore, considering the overall detection effect and computation cost of the framework, we choose $N = 2$ as the value.

H. Impact of Threshold T

The final detection result of the one-class HYBIRD classifier is affected by the threshold T. In order to optimize the detection performance, it is necessary to find the optimal value for the threshold T. Fig. 13 shows the simulation results obtained by our method with different values of T. It can be seen from the figure that when T is increased from 0 to 20, the recall, accuracy, and F-measure values have a significant increase while the precision value has a slight drop. When T is further increased, these four indicators are kept in a high level. After exceeding a certain point of T (i.e., $T = 300$), the values of precision, accuracy, and F-measure are declined while the recall value is increased.

This is because when T is too small, all unknown sigs will be jointly detected by the OC-SVM classifier and the improved SVDD classifier. Although this process may slightly improve the precision value, it may degrade other indicators due to the increased false negatives. After the value of T exceeds a certain point (i.e., $T = 100$), too many fields are only detected by the OC-SVM classifier, which also degrades all indicators. Therefore, it is suggested to set $20 < T < 100$. Within this range, the detection framework can get the best detection results. Of course, this is only the best T value obtained under our data set. In actual applications, relevant personnel can fine tune the T value through testing.

VI. CONCLUSION

The continuous evolution of cyber attacks and the massive use of IoT applications has brought new challenges to anomaly detection in IoT. To address these challenges, a novel anomaly detection framework based on recent advances in EI has been proposed by running sequential clustering and classification on the HTTP traffic, which can effectively and efficiently discover unknown network intrusions. Furthermore, we have presented a data processing method with field detection to eliminate redundant data, which divides the header of HTTP traffic data into multiple fields for detection. This process can accelerate the detection speed and improve the detection performance, as some redundant data are removed for detection. In comparison with two recently proposed anomaly detection methods, the proposed framework has better performance in terms of precision, recall, accuracy, and F-measure, as well as the detection speed. At the same time, the proposed framework can efficiently relieve network congestion and computing pressures of centralized servers, as well as releasing the potential of EI in IoT. Future work is in progress to consider protection-based security schemes, such as continuous authentication, in the proposed framework.

REFERENCES

[1] D. Wang, W. Zhang, B. Song, X. Du, and M. Guizani, "Market-based model in CR-IoT: A Q-probabilistic multi-agent reinforcement learning approach," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 1, pp. 179–188, Mar. 2020.

[2] H. Wang, D. Peng, W. Wang, H. Sharif, H. Chen, and A. Khoynezhad, "Resource-aware secure ECG healthcare monitoring through body sensor networks," *IEEE Wireless Commun.*, vol. 17, no. 1, pp. 12–19, 2010.

[3] H. Wang, D. Peng, W. Wang, H. Sharif, and H.-H. Chen, "Cross-layer routing optimization in multirate wireless sensor networks for distributed source coding based applications," *IEEE Trans. Wireless Commun.*, vol. 7, no. 10, pp. 3999–4009, Oct. 2008.

[4] N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for IoT security based on learning techniques," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2671–2701, 3rd Quart., 2019.

[5] B. Sun, F. Yu, K. Wu, Y. Xiao, and V. C. M. Leung, "Enhancing security using mobility-based anomaly detection in cellular mobile networks," *IEEE Trans. Veh. Technol.*, vol. 55, no. 4, pp. 1385–1396, Jul. 2006.

[6] M. F. Elrawy, A. I. Awad, and H. F. A. Hamed, "Intrusion detection systems for IoT-based smart environments: A survey," *Cloud Comput.*, vol. 7, pp. 21–41, Dec. 2018.

[7] R. Chalapathy and K. Menon, "Anomaly detection using one-class neural networks," 2018. [Online]. Available: arXiv:1802.06360.

[8] W. Yang, W. Zuo, and B. Cui, "Detecting malicious URLs via a keyword-based convolutional gated-recurrent-unit neural network," *IEEE Access*, vol. 7, pp. 29891–29900, 2019.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[10] Y. Zhang, X. Ma, J. Zhang, M. S. Hossain, G. Muhammad, and S. U. Amin, "Edge intelligence in the cognitive Internet of things: Improving sensitivity and interactivity," *IEEE Netw.*, vol. 33, no. 3, pp. 58–64, May/Jun. 2019.

[11] A. Parshin and Y. Parshin, "Investigation of efficient receiving of ultra low power signal for IoT application," in *Proc. 8th Mediterr. Conf. Embedded Comput. (MECO)*, Jun. 2019, pp. 1–4.

[12] P. Duessel, C. Gehl, U. Flegel, S. Dietrich, and M. Meier, "Detecting zero-day attacks using context-aware anomaly detection at the application-layer," *Int. J. Inf. Security*, vol. 16, no. 5, pp. 475–490, 2017.

[13] M. B. Seyyar, F. Ö. Çatak, and E. Gül, "Detection of attack-targeted scans from the apache HTTP server access logs," *Appl. Comput. Informat.*, vol. 14, no. 1, pp. 28–36, 2018.

[14] L. Nie, Y. Li, and X. Kong, "Spatio-temporal network traffic estimation and anomaly detection based on convolutional neural network in vehicular ad-hoc networks," *IEEE Access*, vol. 6, pp. 40168–40176, 2018.

[15] A. Juvonen, T. Sipola, and T. Hämäläinen, "Online anomaly detection using dimensionality reduction techniques for HTTP log analysis," *Comput. Netw.*, vol. 91, p. 46–56, Nov. 2015.

[16] H. Yao, P. Gao, P. Zhang, J. Wang, C. Jiang, and L. Lu, "Hybrid intrusion detection system for edge-based IIoT relying on machine-learning-aided detection," *IEEE Netw.*, vol. 33, no. 5, pp. 75–81, Sep./Oct. 2019.

[17] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1096–1109, May 2018.

[18] T. Omrani, A. Dallali, B. C. Rhaimi, and J. Fattahi, "Fusion of ANN and SVM classifiers for network attack detection," in *Proc. 18th Int. Conf. Sci. Tech. Autom. Control Comput. Eng. (STA)*, Dec. 2017, pp. 374–377.

[19] N. H. Duong and H. D. Hai, "A semi-supervised model for network traffic anomaly detection," in *Proc. 17th Int. Conf. Adv. Commun. Technol. (ICACT)*, Jul. 2015, pp. 70–75.

[20] T. Li, S. Chen, Z. Yao, X. Chen, and J. Yang, "Semi-supervised network traffic classification using deep generative models," in *Proc. 14th Int. Conf. Nat. Comput. Fuzzy Syst. Knowl. Discovery (ICNC-FSKD)*, Jul. 2018, pp. 1282–1288.

[21] J. Wang, L. Yang, J. Wu, and J. H. Abawajy, "Clustering analysis for malicious network traffic," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[22] A. T. Siahmarzkooh, J. Karimpour, and S. Lotfi, "A cluster-based approach towards detecting and modeling network dictionary attacks," *Eng. Technol. Appl. Sci. Res.*, vol. 6, no. 6, pp. 1227–1234, 2016.

[23] A. Feizollah, N. B. Anuar, and R. Salleh, "Evaluation of network traffic analysis using fuzzy c-means clustering algorithm in mobile malware detection," *Adv. Sci. Lett.*, vol. 24, no. 2, pp. 929–932, 2018.

[24] R. R. dos Santos, E. K. Viegas, A. Santin, and V. V. Cogo, "A long-lasting reinforcement learning intrusion detection model," in *Proc. 34th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, Mar. 2020, pp. 1437–1448.

[25] A. H. Basori and S. J. Malebary, "Deep reinforcement learning for adaptive cyber defense and attacker's pattern identification," in *Advances in Cyber Security Analytics and Decision Systems*. Cham, Switzerland: Springer, Jan. 2020, pp. 15–25.

[26] M. Niedermaier, M. Striegel, F. Sauer, D. Merli, and G. Sigl, "Efficient intrusion detection on low-performance industrial IoT edge node devices," 2019. [Online]. Available: arXiv:1908.03964.

[27] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, 2001.

[28] C. She, W. Wen, Z. Lin, and K. Zheng, "Application-layer ddos detection based on a one-class support vector machine," *Int. J. Netw. Security Appl.*, vol. 9, no. 1, pp. 13–24, Jan. 2017.

[29] W. Chen, F. Kong, F. Mei, G. Yuan, and B. Li, "A novel unsupervised anomaly detection approach for intrusion detection system," in *Proc. 3rd Int. Conf. Big Data Security Cloud IEEE Int. Conf. High Perform. Smart Comput. (HPSC) IEEE Int. Conf. Intell. Data Security (IDS)*, May 2017, pp. 69–73.

[30] E. Burnaev and D. Smolyakov, "One-class SVM with privileged information and its application to malware detection," in *Proc. IEEE 16th Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 273–280.

[31] S. Nazir, S. Patel, and D. Patel, "Assessing and augmenting scada cyber security: A survey of techniques," *Comput. Security*, vol. 70, pp. 436–454, Sep. 2017.

[32] B. Liu, Y. Xiao, L. Cao, Z. Hao, and F. Deng, "SVDD-based outlier detection on uncertain data," *Knowl. Inf. Syst.*, vol. 34, no. 3, pp. 597–618, 2013.

[33] H. Danielsson, "Hypertext transfer protocol (http/1.1): Semantics and content," *Faculty Arts Sci.*, vol. 30, no. 4, pp. 595–599, 2017.

[34] M. Flanders, "A simple and intuitive algorithm for preventing directory traversal attacks," 2019. [Online]. Available: arXiv:1908.04502.

[35] H. Gu *et al.*, "Diava: A traffic-based framework for detection of sql injection attacks and vulnerability analysis of leaked data," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 188–202, Mar. 2020.

[36] S. Goswami, N. Hoque, D. K. Bhattacharyya, and J. Kalita, "An unsupervised method for detection of XSS attack," *Int. J. Netw. Security*, vol. 19, no. 5, pp. 761–775, 2017.

[37] H. Xiao, "Birch algorithm and data management in financial enterprises based on dynamic panel gmm test," *Cluster Comput.*, vol. 22, no. 2, pp. 4231–4237, 2019.

[38] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," 2016. [Online]. Available: arXiv:1607.05368.

[39] *Virustotal*. Accessed: Sep. 2020. [Online]. Available: https://www.virustotal.com/

[40] C. Giménez. (2010). *CSIC http Dataset*. [Online]. Available: http://www.isi.csic.es/dataset/