FlowCop: Detecting "Stranger" in Network Traffic Classification

Ningjia Fu*, Yuwei Xu†✉, Jianzhong Zhang†, Rongkang Wang*, Jingdong Xu†,
College of Computer and Control Engineering, Nankai University, Tianjin, China
*{funingjia, wangrongkang}@mail.nankai.edu.cn
†{xuyw, zhangjz, xujd}@nankai.edu.cn

***Abstract*—As the cornerstone of future network research, network traffic classification plays an important role on network management, cyberspace security and quality of service. Recently, many researches have used Machine Learning technologies for traffic classification. Most of them only focus on classifying the samples into predefined classes but ignoring the "strangers". In this paper, we use stranger to represent the traffic not belonging to any predefined application, and propose a novel scheme named FlowCop to achieve stranger detection in network traffic classification. By constructing multiple one-class classifiers, FlowCop can divide testing traffic into N classes and a stranger class. Since samples of stranger class are not required during the training stage, FlowCop works in an inexperienced way to detect strangers, just like the cops searching the crowd for strangers. Besides, for accurate classification and low overhead, a feature subspace algorithm is proposed to select outstanding features for each one-class classifier. To verify the effectiveness of FlowCop, we make contrast experiments on two real-world datasets. The results show that FlowCop can not only identify the predefined traffic flows but also detect the strangers. It outperforms four state-of-the-art approaches on both precision and recall.**

***Index Terms*—Network traffic classification, strange traffic, one-class classifier, feature subspace.**

I. Introduction

In recent years, network traffic classification has been the cornerstone of future network research and attracted more and more attention from academia and industry. On basis of accurate traffic classification, we can filter malicious information, locate the failed devices, and insight into the network state. Thus, it is a very important research to create a healthy network environment. The traditional methods based on port number and deep packet inspection(DPI) become invalid due to the dynamic port and encrypted payload [1]. With the development of artificial intelligence, various of Machine Learning(ML) methods are widely used in the traffic classification based on the flow statistical features.

Studies have shown that 60% of the traffic on the Internet comes from unknown applications [2]. That means, a large number of new applications are emerging on Internet constantly. However, most of the existing ML methods don't consider the traffic may come from a new application that has not been defined in advance, referred as *strangers* in this paper. These methods collect traffic from some predefined applications as the training dataset. Then employ supervised ML techniques to train a classifier. When a testing flow comes, the classifier assigns it to one of the predefined applications. In this way, strangers are misclassified into the predefined classes which results in poor accuracies. To solve this issue, unsupervised clustering methods are applied into the classification stage recent years. Unsupervised clustering methods, by grouping traffic into clusters, can discover strangers according to the mapping gap of clusters and predefined applications. In other words, strangers can be discovered as their locations are not close to any predefined application. One problem with this method is that a large number of strangers are required in the training dataset, which means it is an experiential learning for strange traffic detection.

In this paper, we propose a (N+1)-class classification scheme named FlowCop to tackle this problem. FlowCop can detect strangers in an inexperienced way. The critical idea is to construct one-class classifier for each predefined application. Each classifier can decide whether the traffic belongs to a predefined application or not. In other words, our method aims to identify application A and then application B, instead of distinguish A from B. Thus, the flow rejected by all classifiers can be labeled as stranger. Based on this idea, we only need training data from N predefined applications to build N one-class classifiers, without the requirement of abundant strange flows for the training stage. The contributions of our work are summarized as follows:

1) For the purpose of strange traffic detection, a novel scheme named FlowCop is proposed which can not only identify the flows belonging to predefined applications but also find out the strangers in an inexperienced way.
2) For accurate classification, a feature space algorithm is designed to select the effective features separately for each predefined application. In our algorithm, each feature space is built from low dimension to high dimension to achieve the best performance with the least computation cost.
3) In order to achieve (N+1)-class classification, a novel algorithm is put forward by leveraging multiple one-class classifiers. In our algorithm, each classifier determines whether the testing flow belongs to its corresponding application, and the flow will be labeled as stranger if it is not identified by all the classifiers.

The rest of this paper is organized as follows. Section II reviews the related work on network traffic classification. The methodology of FlowCop is introduced in Section III, which includes its framework, feature subspace and (N+1)-class classification. Section IV describes the experimental results about parameter optimization, analyze the performance of our feature subspace algorithm and compare FlowCop with other four approaches. Finally, the conclusion is given in Section V.

978-1-5386-5156-8/18/$31.00 ©2018 IEEE

II. Related Work

Due to the drawbacks of port-based and payload-based methods in traffic classification, researchers focus on Machine Learning methods recent years. Over the last few years, there have been many papers committed in traffic classification using ML. In what follows, we provide a review of the most representative works using ML methods with the consideration of strange traffic detection.

Traditional traffic classification methods treat the classification task as a N-class classification problem. They define N applications in advance, and then employ supervised methods to train a model with labeled data coming from predefined applications. When the testing traffic arrives, the model can classify it as one of these N applications. These supervised methods include Nearest Neighbors [3], Naive Bayes [4], Decision Tree [5], Neural Network [6], AdaBoost [7], SVM [8] and so on [9]–[12]. In the early work, Williams et al. [4] evaluated 5 supervised algorithms for traffic classification. The results represented that the C4.5 method obtains the highest accuracy, but costs a lot of time. In [5], three supervised ML methods of GP, AdaBoost and C5.0 were investigated on their robustness for VoIP encrypted traffic classification. It proved that C5.0 obtains the best performance on FPR and DR. In recent studies, with more troubles emerging in different classification scenarios, researchers start to employ supervised methods for different purposes. Sinam et al. [10] extracted features from the first few packets of a flow in order to detect VoIP traffic early by Bayesian Belief Network. Considering the validation of traffic, three training issues are highlighted in [9] that should be focused in ML classification. In view of the classification issues introduced by the small-scale dataset, an improved NN classifier is proposed in [3]. All these methods have a typical drawback that without taking strange traffic into account. Supervised methods divide traffic into N predefined classes, but predefined classes cannot cover all existing applications not to mention new emerging applications.

Based on the limitation of single supervised model on multi-class classification, some researchers proposed the "one-class" idea to make supervised methods fit for strange traffic detection."One-class" idea means there are multiple one-class classifiers, and each classifier can decide whether the testing traffic belongs to a predefined application or not. If the testing traffic is not decided as any predefined application, it is labeled as strange traffic. Este et al. [8] did an attempt by constructing multiple one-class SVM classifiers to solve (N+1)-class problem. Their work can assign traffic as stangers but the accuracy is poor.

It is feasible to detect strange traffic by adding unsupervised clustering method into the classification stage, but the works in this direction are scarce. The clustering-based approaches group samples to clusters relying on the similarity between samples, and classify testing data to the class of the nearest cluster. These unsupervised clustering methods include k_means [13], DBSCAN [14], AutoClass [15], Expectation Maximization(EM) [16], etc [17]–[22]. In [18], it is the first time to use unsupervised method for traffic classification. They constructed the classifier by a few labeled data and many unlabeled data. The classifier can map traffic clusters to real applications. However, the huge gap between the cluster number and the application number is hard to address, especially when the labeled training data is very few. A subflow scheme is proposed in [14] that learns the statistical fingerprint for each application respectively, using two clustering methods of k_means and DBSCAN. This scheme is capable to identify strange traffic by the exclusive fingerprints of predefined applications, but the details are not available. Zhang et al. [22] proposed a RTC approach by combining supervised and unsupervised techniques together to address the strange traffic problem. Their method is to use a BoF model to classify flows into (N+1) classes, but this method needs to collect a large number of strange traffic to train the model, which is not intelligent enough.

In conclusion, supervised methods suffer from the strange traffic detection, while unsupervised methods can solve this problem but need abundant strangers in training data. In this paper, we rely on an unsupervised clustering method for feature selection and a integrated scheme by combining supervised and unsupervised methods together for traffic classification. Different from previous works, our model is trained with only predefined flows but can identify strange flows.

III. Proposed Scheme: FlowCop

FlowCop aims to solve the problem of strange traffic detection. The main difference between FlowCop and previous related work is that we can achieve strange traffic detection without strange traffic in training data. To make a full introduction of FlowCop, the following paragraphs are divided into three parts: framework, feature subspace construction and (N+1)-class classification.

A. Framework of FlowCop

There are two stages of FlowCop: the first is feature subspace construction, and the second is (N+1)-class classification, as shown in Fig.1. The ground-truth packets are divided into flows as the training and testing data. The features we use are all flow-level statistic features, including packets number, packet size, interval time and so on. The final output is the classification result covering (N+1) classes.

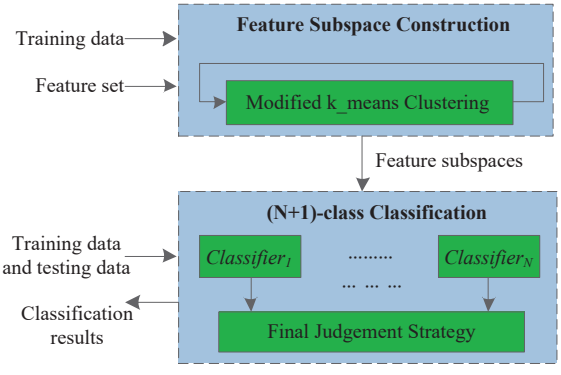


Fig. 1 Framework of FlowCop

In the first stage, with the input of training data and feature set, our feature subspace algorithm can select outstanding features for each applications from the whole feature set. Each predefined application has its unique feature subspace,

which is not the same as others. This algorithm is an iterative algorithm based on the advanced k_means clustering. The feature subspace is obtained through merging features from low dimension to high dimension after each iteration.

In the second stage, we use our (N+1)-class classification algorithm, combining unsupervised k_means method and supervised Nearest Neighbor (NN) method together, to build multiple one-class classifiers. Each classifier can decide whether the testing flow belongs to this application or not. All the decisions from these classifiers are integrated to a final decision covering (N+1) classes. The final decision is determined according to the probability calculated by each classifier. And the traffic rejected by all classifiers can be determined as stranger. In particular, the detection of strange traffic by our algorithm is in an inexperienced way. That means, different from other methods, our FlowCop scheme does not require strange traffic to train the model. FlowCop is trained only using predefined traffic but has the ability to detect strange traffic.

B. Feature Subspace Construction

Feature subspace is composed of some representative features, which are selected from the whole feature set. We use feature subspace instead of the whole feature set to avoid the waste of time and space. Since the representative features of each application are not the same, the feature subspaces are not the same either. For a particular feature subspace, we refer to its corresponding application as *target* class and all the non-corresponding applications as $non-target$ class.

Our feature subspace algorithm is to merge features from low dimension to high dimension by iterating the advanced k_means clustering. In fact, there are many partition-based clustering algorithms available [23], and the k_means algorithm is chosen because of its high speed and simplicity [24]. Some important variables in this algorithm are defined in Tab.I.

Tab. I Some important variables in Algorithm 1

Variable	Description
M	The total number of features
T	The total number of flows
N	The number of predefined applications
$\boldsymbol{O}$	The center of a cluster
$\boldsymbol{F} = \{f_1, f_2, \cdots, f_M\}$	The whole feature set
$\boldsymbol{S} = \{\boldsymbol{s_1}, \boldsymbol{s_2}, \cdots, \boldsymbol{s_T}\}$	The set of training samples
$\boldsymbol{L} = \{l_1, l_2, \cdots, l_T\}$	The labels of samples in $\boldsymbol{S}$
$\boldsymbol{FSS} = \{\boldsymbol{fss_1}, \boldsymbol{fss_2}, \cdots, \boldsymbol{fss_N}\}$	The set of all the feature subspaces
$\boldsymbol{CFSS} = \{\boldsymbol{cfss_1}, \boldsymbol{cfss_2}, \cdots\}$	The set of all current candidate feature subspaces

In our training dataset $\boldsymbol{S}$, each flow $\boldsymbol{s}$ is represented as an M-dimension vector and belongs to one of the predefined applications. Strange traffic does not exist in the training dataset. For any application, we construct 1-deminsion feature subspace at the beginning. We see all features in $\boldsymbol{F}$ as the initial $\boldsymbol{cfss}$ and put them into the candidate set $\boldsymbol{CFSS}$. Each $\boldsymbol{cfss}$ is applied to advanced k_means clustering individually, and the entropy is calculated as the clustering performance of $\boldsymbol{cfss}$. Select $\boldsymbol{cfss}$ that has the entropy smaller than the average of all entropies, then send the selected $\boldsymbol{cfss}$ to the merging process. In the merging process, for all selected $\boldsymbol{cfss}$, any two of them are merged as a new higher-dimension $\boldsymbol{cfss}$. Then the set $\boldsymbol{CFSS}$ is updated by these new $\boldsymbol{cfss}$. Iterate the above steps until the smallest entropy is no longer reduced, or the dimension of $\boldsymbol{cfss}$ reaches the highest. When the merging process ends, we choose the $\boldsymbol{cfss}$ with the smallest entropy as the final $\boldsymbol{fss}$ of the target application. Our feature subspace algorithm is summarized with the pseudocode in Algorithm 1.

Algorithm 1: Feature subspace algorithm

input: $\boldsymbol{S}$, $\boldsymbol{F}$, $\boldsymbol{L}$, k, M, N
output: $\boldsymbol{FSS}$

```
for App_i, i = 1, 2, ..., N do:
    initialize CFSS, CFSS = {cfss_1, cfss_2, ..., cfss_M}, cfss_j ← f_j
    dimension = max_{cfss∈CFSS} |cfss|
    initialize bestE: bestE = ∞
    initialize bestfss: bestfss = ∅
    change the label of s ∈ App_i as target, others as non − target.
    while dimension ≤ M do:
        for each cfss in CFSS do:
            generate S' based on S and the features in cfss
            C = k_means(S', k)
            calculate E(C) by formula (3)
        end for
        minE = min(E(C))
        if minE ≥ bestE do:
            break;
        else do:
            update bestE : bestE = minE
            update bestfss: bestfss = the cfss corresponding to minE
            CFSS' = {cfss|cfss ∈ CFSS, entropy_cfss < avg(E)}
            merge all cfss ∈ CFSS' in pairs as the new cfss
            update CFSS with the new cfss
            update dimension : dimension = max_{cfss∈CFSS} |cfss|
        end if
    end while
    fss_i = bestfss
    add fss_i into FSS
end for
return FSS
```

The merging step is the kernel of Algorithm1. In theory, if samples show a high separation between different classes in a low-dimension space, it is possible to show a better separation in the higher-dimension space after merging two such low-dimension spaces together.

In our algorithm, the entropy is used to evaluate the quality of clusters. Entropy can describe the degree of confusion of the grouped clusters. A small entropy means the separation between *target* class and $non-target$ class is clear. For a clustering result $\boldsymbol{C} = \{\boldsymbol{c_1}, \boldsymbol{c_2}, \cdots, \boldsymbol{c_k}\}$, the entropy of cluster $\boldsymbol{c_i}$ is defined as:

$$E(\boldsymbol{c}_j) = -\frac{1}{log2}[p_{1j} log\, p_{1j} + p_{2j} log p_{2j}] \tag{1}$$

P_{ij} is the probability that the cluster $\boldsymbol{c_i}$ belongs to class i and it is calculated by formula (2). $|\boldsymbol{B_{ij}}|$ is the number of elements in the set $\boldsymbol{B_{ij}}$. Because there are only two classes (*target* and $non-target$), the probabilities of them satisfy the relationship $p_{2j} = 1 - p_{1j}$.

$$p_{ij} = \frac{|\boldsymbol{B_{ij}}|}{|\boldsymbol{c_j}|}, \quad \boldsymbol{B_{ij}} = \{\boldsymbol{s} | \boldsymbol{s} \in \boldsymbol{c_j}, \boldsymbol{s} \in class\ i\} \tag{2}$$

The entropy of the clustering result $\boldsymbol{C}$ is defined as $E(\boldsymbol{C})$, which is obtained by the weighted sum of $E(\boldsymbol{c_j})$ as shown in formula (3). We use a threshold of entropy to judge which $\boldsymbol{cfss}$ can continue to the next merging phase, and this threshold is set as the average of entropies during once merging operation.

$$E(\boldsymbol{C}) = \sum_{j=1}^{k} [\frac{|\boldsymbol{c_j}|}{n} E(\boldsymbol{c_j})] \tag{3}$$

Advanced k_means algorithm: Advanced k_means algorithm is the basis of Algorithm1. Different from the original k_means algorithm, the advanced k_means algorithm has three improvements: the initialization of cluster centers, the update of cluster centers, and the convergence conditions.

1) The initialization of cluster centers: The strategy of initializing cluster centers is to add a random vector $\boldsymbol{\varepsilon}$ to the average value of all training samples. Each flow is represented as a M-dimension vector, like $\boldsymbol{s_i} = (s_{i1}, s_{i2}, \cdots, s_{iM})$, so T flows can be represented as a matrix $\boldsymbol{S}$:

$$\boldsymbol{S} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1M} \\ s_{21} & s_{22} & \cdots & s_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ s_{T1} & s_{T2} & \cdots & s_{TM} \end{bmatrix}$$

We define $\bar{\boldsymbol{S}} = (\mu_1, \mu_2, \cdots, \mu_M)$, where $\mu_j = \frac{1}{T}\sum_{j=1}^{T} s_{ij}$, representing the average value of all training samples in dimension j. For each cluster, the initial cluster center is obtained by formula (4).

$$\boldsymbol{O}^{(0)} = \bar{\boldsymbol{S}} + \boldsymbol{\varepsilon} \tag{4}$$

$\boldsymbol{\varepsilon}$ is a M-dimension random vector represented as $(\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_M)$, and each element is calculated by formula (5). $\mathcal{N}(0,1)$ is a function that can generate a random number satisfying the normal distribution. Based on the 3σ rule of normal distribution, $\mathcal{N}(0,1)$ is divided by 3 to make the random number between 0 and 1.

$$\varepsilon_j = \frac{\mathcal{N}(0,1)}{3} \cdot (\max_{1 \le i \le T}\{s_{ij}\} - \min_{1 \le i \le T}\{s_{ij}\}) \tag{5}$$

2) The update of cluster centers: For each training sample $\boldsymbol{s}$, calculate the L2 norm between $\boldsymbol{s}$ and all cluster centers $\boldsymbol{O}$. *cid* records which cluster center is closest to $\boldsymbol{s}$, and w records the distance from $\boldsymbol{s}$ to the closest cluster *cid*.

$$cid = \arg\min_{j} \|\boldsymbol{s} - \boldsymbol{O_j^{(t)}}\|_2, \quad 1 \le j \le k \tag{6}$$

$$w = \|\boldsymbol{s} - \boldsymbol{O_{cid}^{(t)}}\|_2 \tag{7}$$

Then we update all cluster centers by formula (8). For cluster j, we define a set $\boldsymbol{B} = \{b | \boldsymbol{s_b} \in \boldsymbol{c_j}\}$, and w_i is treated as the weight for each $\boldsymbol{s_i}$.

$$\hat{\boldsymbol{O}}_{\boldsymbol{j}}^{(t+1)} = \sum_{i \in B} (w_i \cdot \boldsymbol{s_i}) / \sum_{i \in B} w_i \tag{8}$$

Sometimes the cluster centers may oscillate and cannot converge. This phenomenon mainly occurs when the number of samples is small and samples have a sparse distribution. We use a variable named *damping* to solve the oscillation problem. The method is to take the last cluster centers $\boldsymbol{O}^{(t)}$ into account when update the new cluster centers $\boldsymbol{O}^{(t+1)}$, and the variable *damping* is added as the weight. The formula (8) is optimized as formula (9). The *damping* determines the convergence rate and we set $damping = 0.1$ in terms of the empirical study.

$$\boldsymbol{O}^{(t+1)} = (1 - damping) * \hat{\boldsymbol{O}}^{(t+1)} + damping * \boldsymbol{O}^{(t)} \tag{9}$$

3) The convergence conditions: We define the cost function J as formula (10) to record the changes of cluster centers after each iteration. J is obtained by the L2 norm of the cluster centers before and after the update, which indicates the offset distance of the cluster centers. We set a small threshold for J to determine whether the iteration should be terminated or not. Here it is set to be 0.1 based on the empirical study.

$$J = \sum_{i=1}^{k} \|\boldsymbol{O_i^{(t+1)}} - \boldsymbol{O_i^{(t)}}\|_2 \tag{10}$$

C. (N+1)-Class Classification

Based on the feature selection results in Part B, the (N+1)-class classification algorithm is proposed to build our classification scheme. There are two steps to achieve (N+1)-class classification. Firstly, multiple one-class classifiers are constructed. Secondly, a final decision is given by integrating all these outputs together. Through these two steps, our FlowCop scheme can achieve inexperienced strange traffic detection. The principle is explained in detail as follows.

When a classifier gives a decision as $non-target$, it means the testing sample fails to group with the target training samples in the optimal feature subspace. Meanwhile, this testing sample may also fails to group with the non-target training samples, because it may come from an unknown application so it cannot group with any existed application defined in our system. Labeled as $non-target$ only means this sample does not belong to the *target* application but no matter it belongs to a $non-target$ application or not. If no classifier labels it as *target*, this testing sample is highly posible to come from a strange application. Briefly, the traffic rejected by all pre-dfined applications can be determined as strange traffic.

1) One-class classifier: Our one-class classifier combines k_means and NN (nearest neighbor) together to classify a testing flow as *target* or $non-target$. In the feature subspace algorithm, samples of different classes can be separated well by k_means in the optimal feature subspace, but the classification performance only by k_means is not reliable. Using k_means to classify traffic means the testing sample is assigned to the nearest cluster. However, the class of a cluster is defined as the major class of all samples in this cluster. It means not all samples in this cluster belong to the major class. That is to say, the cluster's class is not entirely reliable and cannot be used as a credible rule to divide traffic. Based on this thinking, we combine NN algorithm and k_means algorithm together to make the results more reliable. We select several nearest training samples close to the testing sample, and calculate a probability as the rule to divide traffic. "Nearest" means the most similar. Using several similar samples to replace the similar cluster can be more reliable. In addition, we give these nearest training samples different weights to strengthen the reliability of results.

Some important variables in the algorithm are defined in Tab.II. And the pseudocode is shown in Algorithm 2. Firstly, we cluster training samples by advanced k_means using the feature subspace. Then, we cut off some clusters that satisfy $d(\boldsymbol{u}, \boldsymbol{O}) \ge r_{max}$. In the remaining clusters, we select the nearest k' training samples close to the testing sample. Based on the labels of these k' training samples, we calculate the ownership P_{target} and $P_{non-target}$ by formula (11) and (12). If P_{target} is

larger than $P_{non-target}$, the testing sample is recongnized as *target* and return P_{target}, otherwise as $non-target$ and return 0. These outputs are useful for the final decision described in Part.2.

Tab. II Some important variables in Algorithm 2

Variable	Description
$\boldsymbol{C}=\{\boldsymbol{c_1},\boldsymbol{c_2},\cdots,\boldsymbol{c_k}\}$	The clusters returned by k_means
$\boldsymbol{S}=\{\boldsymbol{s_1},\boldsymbol{s_2},\cdots\}$	The set of all training samples
$\boldsymbol{L}=\{l_1,l_2,\cdots\}$	The labels of training samples
$\boldsymbol{U}=\{\boldsymbol{u_1},\boldsymbol{u_2},\cdots\}$	The set of all testing samples
$\boldsymbol{R}=\{r_1,r_2,\cdots\}$	The classification results of testing samples
r_{max}	The max radius of a cluster
d_{max}	The max distance from $d(\boldsymbol{s_i},\boldsymbol{u})$, $1\le i\le k'$

We define set $\boldsymbol{B}=\{b|\boldsymbol{s_b}\in App_i,\ \boldsymbol{s_b}\in\{\boldsymbol{s_1},\boldsymbol{s_2},\cdots,\boldsymbol{s_{k'}}\}\}$ and its complement set $\bar{B}=\{b|\boldsymbol{s_b}\notin App_i,\ \boldsymbol{s_b}\in\{\boldsymbol{s_1},\boldsymbol{s_2},\cdots,\boldsymbol{s_{k'}}\}\}$. Each sample in $\boldsymbol{B}$ and $\bar{B}$ has a weight $(\lambda+\eta)$.

$$P_{target}=\frac{1}{k'}\sum_{i\in B}[\lambda(\boldsymbol{s_i})+\eta(\boldsymbol{s_i})] \tag{11}$$

$$P_{non-target}=\frac{1}{k'}\sum_{i\in \bar{B}}[\lambda(\boldsymbol{s_i})+\eta(\boldsymbol{s_i})] \tag{12}$$

The factor λ is calculated based on the training sample's location in the cluster as formula (13). The factor η is calculated based on the distance between testing sample and training sample as formula (14).

$$\lambda(\boldsymbol{s})=(r_{\max}-d(\boldsymbol{s},\boldsymbol{O}))/r_{\max} \tag{13}$$

$$\eta(\boldsymbol{s})=(d_{\max}-d(\boldsymbol{s},\boldsymbol{u}))/d_{\max} \tag{14}$$

Here, the weight $(\lambda+\eta)$ is used to solve the class skew problem, which means the classification result is easily influenced by the large-scale class. In the traditional NN algorithm, the final decision is obtained by the vote rule, and each training sample has one vote. Thus, testing sample from a small-scale class is more easily misclassified to a large-scale class due to its strength on sample size. In our approach, there are only two classes (*target* and $non-target$) in a single classifier, and the sample number of $non-target$ class is much larger than that of *target* class. Therefore, the class skew phenomenon cannot be ignored. In formula (11) and (12), instead of directly counting the classes of k' nearest neighbors, we calculate a weighted summation of $(\lambda+\eta)$ by taking two distances namely $d(\boldsymbol{s},\boldsymbol{O})$ and $d(\boldsymbol{s},\boldsymbol{u})$ into account. The training sample located at the edge of the cluster has a weak weight for classification, and the training sample far from the testing sample has a weak weight too. By this way, we can ensure k' nearest training samples have different influence in classification stage so as to mitigate the effect of class skew.

2) (N+1)-class classification: As each classifier can identify a certain application, there are totally N outputs after the first step. Therefore, a strategy is needed to merge them together to obtain a final decision covering (N+1) classes.

As mentioned in the last part, one-class classifiers can decide testing traffic as *target* or $non-target$. If the decision is *target*, it gives P_{traget} as the output, otherwise it outputs 0. The strategy we design is as follows. For a testing sample, if the outputs of N classifiers are all 0, we decide this testing sample as strange traffic. If there is only one output larger than 0, this sample is assigned to the corresponding application class. If there are several outputs that are larger than 0, we decide this testing sample as the application class that has a maximum P_{target}. The pseudocode is summarized in Algorithm3.

Algorithm 2: One-class classification algorithm

input: $\boldsymbol{S},\boldsymbol{U},\boldsymbol{L},\boldsymbol{FSS},k'$
output: $\boldsymbol{R}$

for $classifier_i$ **do**:
  generate $\boldsymbol{S'}$ based on $\boldsymbol{S}$ and $\boldsymbol{fss_i}$
  generate $\boldsymbol{U'}$ based on $\boldsymbol{U}$ and $\boldsymbol{fss_i}$
  $\{\boldsymbol{C},\boldsymbol{O},r_{max},r_{min}\}=k\_means(\boldsymbol{S'},k)$
  for each $\boldsymbol{u_i}$ in $\boldsymbol{U}$ **do**:
    initialize ***delete***: ***delete*** $=\emptyset$
    for j from 1 to k **do**:
      calculate $d(\boldsymbol{u_i},\boldsymbol{O}(j))$
      if $d(\boldsymbol{u_i},\boldsymbol{O}(j))>r_{max}(j)$ **do**:
        add j into the set ***delete***
      end if
    end for
    if $|\boldsymbol{delete}|=k$ **do**:
      $\boldsymbol{R}(i)=0$
    else do:
      remove clusters from $\boldsymbol{C}$ based on ***delete***, the remaining clusters are $\boldsymbol{C'}$
      for all $\boldsymbol{s}\in\boldsymbol{C'}$, calculate the distance $d(\boldsymbol{u_i},\boldsymbol{s})$
      sort all $d(\boldsymbol{u_i},\boldsymbol{s})$ and get the nearest $\{\boldsymbol{s_1},\boldsymbol{s_2},\cdots,\boldsymbol{s_{k'}}\}$
      calculate $\{(\lambda+\eta)_1,(\lambda+\eta)_2,\cdots,(\lambda+\eta)_{k'}\}$ by formula (13) and (14)
      calculate P_{target} and $P_{non-target}$ by formula (11) and (12)
      if $P_{target}<P_{non-target}$ **do**:
        $\boldsymbol{R}(i)=0$
      else do:
        $\boldsymbol{R}(i)=P_{target}$
      end if
    end if
  end for
return $\boldsymbol{R}$
end for

IV. Experiments

In this section, we do some experiments to get the optimal parameters and evaluate the effective and robustness of FlowCop. For a comparison, we also implement four existing classification methods. Naive Bayes and C4.5 Decision Tree [4], which represent the traditional Machine Learning models, are adopted due to the good performance in traffic classification. The third is the RTC model proposed by Zhang [22]. RTC can assign traffic into (N+1) classes. The first step of RTC is the stranger discovery. They use a semi-supervised approach to cluster the training dataset including labeled and unlabeled traffic, so that they can discover the strange traffic among the unlabeled traffic and then use them to train the classifier. But in this way, the training data they use needs to cover strangers, which means it isn't inexperienced. The last one is the one-class SVM model [8], which creates multiple one-class SVM classifiers to solve multi-class classification problem as well as the strange traffic problem. Their idea is the same as us but they build all classifiers in the same feature space. Besides, for the reliability of conclusions, all the experiments are repeated 100 times to get the average results.

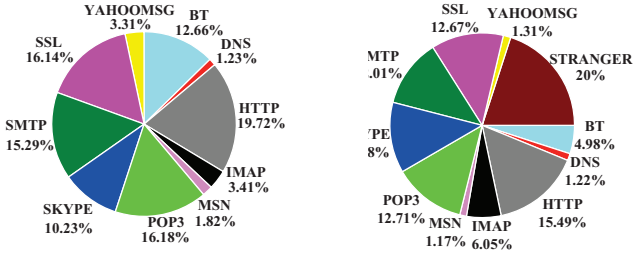
Algorithm 3: (N+1)-class classification algorithm

input: $classifier_1, classifier_2, \cdots, classifier_N, \boldsymbol{U}$
output: R

```
for each u_i in U do:
    for j from 1 to N do:
        R_temp(j) = classifier_j(u_i)
    end for
    get B = {r_temp | r_temp ≠ 0, r_temp ∈ R_temp}
    if |B| = 0 do:
        R(i) = stranger
    else if |B| = 1 do:
        R(i) = {App_l | r_temp ∈ App_l}
    else do:
        P_max = max{r_temp | r_temp ∈ B}
        R(i) = {App_l | P_max ∈ App_l}
    end if
end for
return R
```

A. Dataset

For our experiments, we use our own dataset covering 10 predefined applications including BT, DNS, HTTP, IMAP, MSN, POP3, SKYPE, SMTP, SSL, YAHOOMSG. The traces are captured from our testbed, at the edge of our campus network. We capture great number of ground truth application traces using the method of [25] from 2016 to 2017. In addition, the public dataset provided by MAWI Working Group [26] is used to extend our dataset. The MAWI Working Group captured daily traces at the transit link of WIDE to the upstream ISP since 2000 to 2017, and still update the dataset constantly. Their traffic traces are collected at 6 sampling points, from samplepoint-A to samplepoint-F. We use the dataset of samplepoint-F with the time from 2015 to 2017.



(a) Training dataset (b) Testing dataset

Fig. 2 Class distribution of the dataset

The total dataset we use has a size of 300GB, and it is divided into two disjointed parts: training dataset and testing dataset. Totally 54,000 training flows are used in experiments coming from all 10 predefined applications. For testing flows, besides 40,000 flows covering all predefined applications, there are 10,000 additional flows coming from other applications such as RAZOR, XMPP, EBUDDY, etc. These applications serve as new applications to test the classifier's ability to detect strange traffic. Fig.2 shows the distribution of traffic classes in our training dataset and testing dataset.

B. Parameter Optimization

In our algorithm, there are two important parameters need to be adjusted to achieve a best classification performance. One is k representing the number of clusters, the other is k' representing the number of nearest training neighbors.

1) The optimization of parameter k : Parameter k represents the number of clusters we want to group in the advanced k_means algorithm. We optimize the parameter k for each application. The value of k changes from 10 to 2010 with the increment of 50. For each k, we run Algorithm1 to get the optimal feature subspace and record the smallest entropy as the performance of this k.

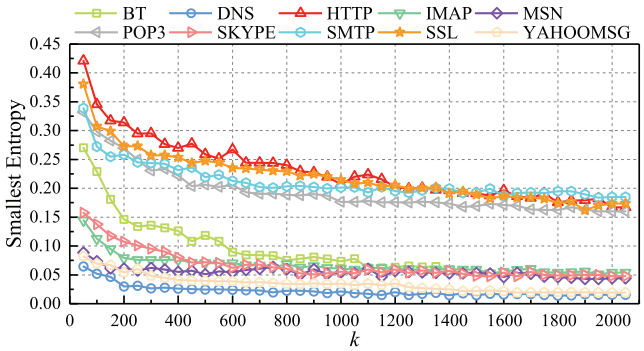


Fig. 3 Classification performance with variable k

In Fig.3, the results show the overall trend of each curve decreases first and then tends to be stable. Basically, k = 10 to 210 is the most declining range, and the curve tends to be flat gradually after k = 210 . This is because, as k is small initially, only a few clusters are generated and each cluster contains a large number of samples. In this case, the classes of samples in a cluster are messy, resulting in a low purity and a high entropy. As k increases, the number of clusters increases and samples at a slight distance are grouped into different clusters, so that the samples grouped in a cluster are very similar, resulting in a small entropy. The k where the entropy starts to be stable is the optimal k we are looking for. To find the optimal k, we calculate the difference between two adjacent points on the curve. If 5 consecutive distances are all less than 0.01, we see the curve is stable and the first point of these 5 distances is detrmined as the optimal k. Based on the experimental results, the optimal k determined for each application are in Tab.III.

Tab. III Optimal k and k' for each application

ID	Application	Optimal k	Optimal k'
1	BT	560	4
2	DNS	160	7
3	HTTP	960	7
4	IMAP	210	4
5	MSN	210	4
6	POP3	960	7
7	SKYPE	810	14
8	SMTP	410	4
9	SSL	1860	7
10	YAHOOMSG	410	11

2) The optimization of parameter k' : In the classification stage, we have to select k' nearest training samples for each testing sample to calculate P_{target} and $P_{non-target}$ that play an important role for the final decision. The value of k' decides how many training samples that can influence the final decision when we classify a testing sample. As each one-class classifier is individually, we select the optimal k' for each classifier.

Fig.4 records the accuracy as k' changes from 1 to 85 with the increment of 3. When $k' > 85$, the accuracy decreases

slowly, so here we only discuss the results when $k' \leq 85$. We can see that as k' increases, most applications such as BT, HTTP, and SSL show a trend that the accuracy firstly increases and then decreases dramatically, and finally decreases slowly. The dramatic variation of accuracy occurs when $k' \in [0, 20]$. As $k' > 20$, the accuracy tends to be declining slowly. In this case, the optimal k' we are looking for is determined as the point where the accuracy is at the peak. The optimal k' selected for each application is shown in Tab.III. We can see the optimal k' for each application is less than 15, much smaller than the number of training samples per application.

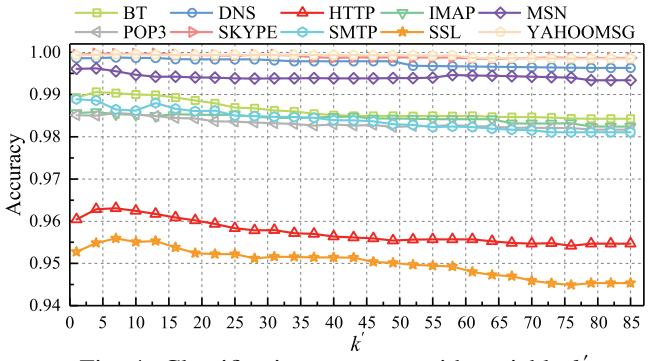


Fig. 4 Classification accuracy with variable k'

C. Performance of Feature Subspace

The whole feature set we use in this paper includes 20 flow statistical features, as listed in Tab.IV. They are chosen because the good performance in traffic classification in previous studies. They can be divided into two groups: size-relevant and time-relevant. Size-relevant means the packet size and the byte size. Time-relevant means the inter packet time. They are all numerical variables and are normalized using the standard z-score algorithm [14].

For each application, we use the feature subspace obtained by Algorithm1 and randomly generate 20 extra feature subsets in different dimensions from the whole feature set to classify the testing dataset by Algorithm2. Becides the accuracy of the feature subspace, we also record the maximum, minimum and average accuracy of the extra 20 feature subsets as the comparison, as shown in Fig.5.

Tab. IV Statistical flow-level features

Type	Feature Description	Number
size-relevant	Min., Max., Mean and Std Dev. of packets number, bytes number, packet size	12
time-relevant	Min., Max., Mean and Std Dev. of interval time	8

According to the experimental results, most of the classifiers achieve the highest accuracy in the selected feature subspace(FSS). It shows that compared with other feature subsets, FSS has a better classification performance, which proves the optimality of the feature subspace selected by our algorithm. In particular, YAHOOMSG has a higher accuracy on MAX than FSS. One reason is that there may be an occasional feature subset performing better than FSS due to the randomness of our algorithm. In fact, FSS is obtained through the merging step from low dimension to high dimension. The algorithm does not traverse all feature subsets in all dimensions, and the selected FSS is not necessarily global optimal, so it is normal that a certain feature subset preforms better than FSS. But this case is rare, and there is only a small difference on accuracy between FSS and MAX, indicating that FSS we selected can regarded as "optimal" to a large extent. Besides, instead of traversing all feature subsets to select a global optimal feature subspace, our method can save a lot of time as well as ensuring the quality of FSS, which verifies the rationality and correctness of our algorithm.

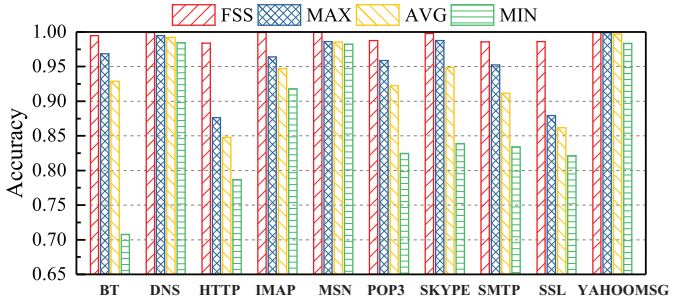


Fig. 5 Classification performance of feature subspaces

Because of the improvements made in Section III-B, the convergence speed of our advanced k-means algorithm is increased by 20%. Besides, the clustering accuracy is also improved by 5%. Owing to space constraints, we omit the details of our experiments here.

D. Performance of One-class Classifier

To evaluate the performance of one-class classifiers, we compare the classification accuracy of FlowCop and one-class SVM [8]. One-class SVM is chosen because it has the same principle as our method, and the other 3 methods don't have the ability of one-class classification. The implement of one-class SVM is in terms of the description in [8], and the parameter is set as the optimal value in [8].

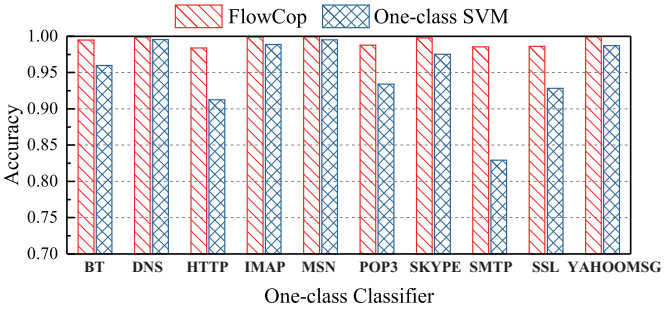


Fig. 6 FlowCop's classifiers VS one-class SVM's

In Fig.6, the results show that FlowCop's classifier has better classification accuracy than SVM's for all applications, and the gap of accuracy is quite large in some applications, which fully proves the great performance of FlowCop. Analyze the causes as follows. Firstly, one-class SVM cannot produce a discriminative boundary with a miscellaneous training dataset. The principle of SVM is to construct a boundary in the feature space, which can separate samples of two classes well. In this paper, two classes are defined as *target* and $non-target$, but samples actually come from many applications. Thus, it is difficult to separate them well through a single boundary. Secondly, SVM method build the classifiers in the same feature space. It does not have the step of feature selection, so it is easy to be affected by the noisy features, and the boundary

constructed in a high-dimension feature space is less effective. Thirdly, its capability of strange traffic detection is limited as strange traffic does not exist in training data.

E. Performance of (N+1)-class Classification

We evaluate our FlowCop scheme comparing with other four state-of-art methods in this part. For each class, we record the precision and recall as the metrics [27]. Besides, the global overall accuracy is also counted as a metric. These three metrics are defined as below:

$$precision = TP/(TP + FP) \tag{15}$$

$$recall = TP/(TP + FN) \tag{16}$$

$$overall\ \ accuracy = \sum_{i=1}^{N+1} TP_i / \sum_{i=1}^{N+1} (TP_i + FN_i) \tag{17}$$

where, True Positives (TP) is the number of class i's samples identified as class i; False Positives (FP) is the number of other class's samples identified as class i; False Negatives (FN) is the number of class i's samples identified as other class.

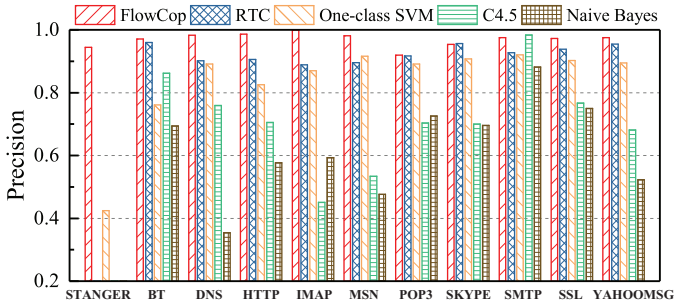


Fig. 7 Precision of five competing methods

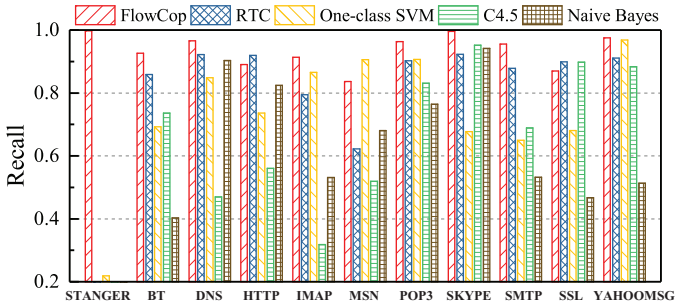


Fig. 8 Recall of five competing methods

Fig.7 and Fig.8 shows the precision and recall of 11 classes for 5 competing methods. Two conclusions can be drew according to these two figures. 1) For 10 predefined applications, FlowCop performs best on both precision and recall. Our algorithm takes full account of the distribution of samples, and makes full use of the feature subspace. Due to this, FlowCop has a great power on the classification of N predefined applications. RTC is the second and also has a good performance. On HTTP and SSL, RTC's recall even outweighs FlowCop. The BoF model of RTC can extract the correlation information between flows, which enhances the accuracy of the classification results. For one-class SVM method, the single boundary that constructed on the multi-class dataset cannot divide *target* and $non-target$ traffic effectively, so the percision and recall of one-class SVM are worse than FlowCop and RTC. The remaining two are traditional Machine Learning methods. C4.5 performs slightly better than Naive Bayes, but the effects of these two are worse than other methods. 2) For strangers, FlowCop has a great capability of strange traffic detection that far superior to the other four methods. In fact, only FlowCop and one-class SVM can achieve strange traffic detection. Owing to the "one-class" idea, SVM method can identify a small number of strangers in the inexperienced way, shown as the slight precision and recall on stranger class. RTC needs strange traffic in training dataset to build (N+1)-class classification model, so it cannot detect strangers in the inexperienced way, which is the major drawback of RTC.

For the overall accuracy, FlowCop reaches the highest accuracy, followed by RTC. As RTC cannot achieve the inexperienced strange traffic detection, strangers in testing dataset are misclassified to the predefined applications, which causes a low overall accuracy. Considering three metrics of overall accuracy, precision and recall, FlowCop has the best performance on 10 predefined classes and the stranger class, and it can identify strange traffic in the inexperienced way.

Finally, we calculate the average running speed of those methods. One-class SVM gets the slowest speed of 2.3×10^3 flows per second(fps), and RTC obtains 3.1×10^3 fps. The results of C4.5 and Naive Bayes are 1.02×10^4 fps and 1.17×10^4 fps respectively. Due to structure of multiple classifiers, the average speed of FlowCop is 2.8×10^3 fps. Although it is slower than other three methods, it is still faster than one-class SVM. Therefore, FlowCop obtains the highest precision and recall with an acceptable speed. Moreover, considering the structure of Flowcop, the construction and classification process of each classifier can be paralleled due to its independence, which can be a follow-up research direction to increase the speed of Flowcop.

F. Impact of Training Data Size

In this part, we test the robustness and stability of FlowCop by increasing the training data size. The size of training data is increased from 1,000 to 53,000 with increment of 2,000. Other 4 methods are also tested as the comparison.

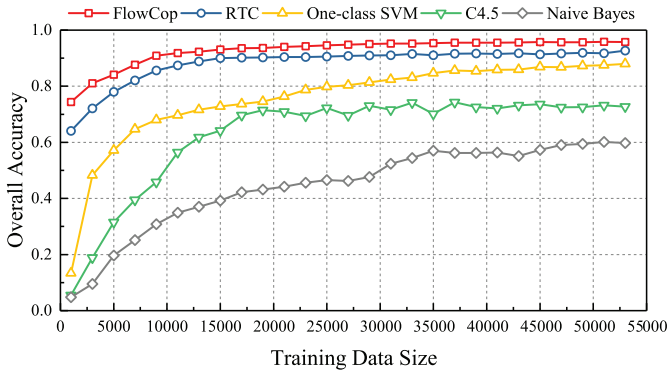


Fig. 9 Impact of training data size

Fig.9 shows the overall accuracy of 5 competing methods as the training data size increases. For each method, the overall accuracy first increases and then stabilizes. For FlowCop, RTC and one-class SVM, the accuracy rises greatly when the training data size is from 1,000 to 10,000. For C4.5 and Naive Bayes, the accuracy rises at a further range, from 1,000 to

20,000. In the whole change of training data size, we can see the curve of FlowCop is always at the top, which shows the performance of FlowCop is better than that of other four methods. It fully proves the high effectiveness of FlowCop. The performance of RTC is only slightly behind FlowCop, and the difference is reflected in the strange traffic detection which is not inexperienced for RTC. One-class SVM enables strange traffic detection, but the detection is less accurate because it fails to establish an effective boundary with multiple classes of training data.

For the jitter of the curve in Fig.9, the curve of FlowCop is very stable with few jitter points, which proves its great robustness. Similarly, RTC also has a good robustness. However, for the remaining three methods, there are more jitter points in the curves. After the curve tends to be stable, there are still some waves in the curve, which means a poor stability.

Considering the gap between the lowest and the highest accuracy, the gap of FlowCop is the smallest, followed by RTC. For FlowCop, the overall accuracy always maintains at the highest level regardless of the training data size, together with the small gap, indicating that FlowCop can achieve a great performance on classification no matter the training data is more or less. These results further confirm the robustness of the proposed FlowCop scheme.

V. Conclusion

This paper aims to address the problem of strange traffic detection. To achieve (N+1)-class classification, we propose a traffic classification scheme named FlowCop. FlowCop is not only effective for predefined applications, but also can achieve inexperienced detection for strange traffic. Firstly, we propose a feature subspace algorithm. This algorithm can select outstanding features for each application. Secondly, we propose a (N+1)-class classification algorithm that can divide traffic into (N+1) classes, of which the strange traffic detection is inexperienced. We do many experiments comparing FlowCop with other methods, and the results confirm that FlowCop has a great classification performance for both predefined traffic and strange traffic, as well as the great robustness and stability.

VI. Acknowledgement

This work was supported by the National Natural Science Foundation of China (NO. 61702288), the Natural Science Foundation of Tianjin in China (No. 16JCQNJC00700) and the Fundamental Research Funds for the Central Universities.

References

[1] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[2] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT conference*. ACM, 2008, p. 11.

[3] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Transactions on Parallel and Distributed systems*, vol. 24, no. 1, pp. 104–117, 2013.

[4] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[5] R. Alshammari and A. N. Zincir-Heywood, "Identification of voip encrypted traffic using a machine learning approach," *Journal of King Saud University-Computer and Information Sciences*, vol. 27, no. 1, pp. 77–92, 2015.

[6] Z. Nascimento, D. Sadok, S. Fernandes, and J. Kelner, "Multi-objective optimization of a hybrid model for network traffic classification by combining machine learning techniques," in *International Joint Conference on Neural Networks*. IEEE, 2014, pp. 2116–2122.

[7] R. Alshammari and A. N. Zincir-Heywood, "How robust can a machine learning approach be for classifying encrypted voip?" *Journal of Network and Systems Management*, vol. 23, no. 4, pp. 830–869, 2015.

[8] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for tcp traffic classification," *Computer Networks*, vol. 53, no. 14, pp. 2476–2490, 2009.

[9] H. A. H. Ibrahim, O. R. A. Al Zuobi, M. A. Al-Namari, G. MohamedAli, and A. A. A. Abdalla, "Internet traffic classification using machine learning approach: Datasets validation issues," in *Basic Sciences and Engineering Studies*. IEEE, 2016, pp. 158–166.

[10] T. Sinam, N. Ngasham, P. Lamabam, I. T. Singh, and S. Nandi, "Early detection of voip network flows based on sub-flow statistical characteristics of flows using machine learning techniques," in *Advanced Networks and Telecommuncations Systems (ANTS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1–6.

[11] Y.-n. Dong, J.-j. Zhao, and J. Jin, "Novel feature selection and classification of internet video traffic based on a hierarchical scheme," *Computer Networks*, vol. 119, pp. 102–111, 2017.

[12] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 50–60.

[13] A. Kumar, J. Kim, S. C. Suh, and G. Choi, "Incorporating multiple cluster models for network traffic classification," in *LCN, 2015 IEEE 40th Conference on*. IEEE, 2015, pp. 185–188.

[14] G. Xie, M. Iliofotou, R. Keralapura, M. Faloutsos, and A. Nucci, "Subflow: Towards practical flow-level traffic classification," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2541–2545.

[15] Z. Zhang, Z. Zhang, P. P. Lee, Y. Liu, and G. Xie, "Toward unsupervised protocol feature word extraction," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 10, pp. 1894–1906, 2014.

[16] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *International Workshop on Passive and Active Network Measurement*. Springer, 2004, pp. 205–214.

[17] P. Zhu, W. Zhu, Q. Hu, C. Zhang, and W. Zuo, "Subspace clustering guided unsupervised feature selection," *Pattern Recognition*, vol. 66, pp. 364–374, 2017.

[18] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, 2007.

[19] X. Zhao, W. Deng, and Y. Shi, "Feature selection with attributes clustering by maximal information coefficient," *Procedia Computer Science*, vol. 17, pp. 70–79, 2013.

[20] S. Bandyopadhyay, T. Bhadra, P. Mitra, and U. Maulik, "Integration of dense subgraph finding with feature clustering for unsupervised feature selection," *Pattern Recognition Letters*, vol. 40, pp. 104–112, 2014.

[21] W. Lu and L. Xue, "A heuristic-based co-clustering algorithm for the internet traffic classification," in *International Conference on Advanced Information NETWORKING and Applications Workshops*. IEEE, 2014, pp. 49–54.

[22] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *TON*, vol. 23, no. 4, pp. 1257–1270, 2015.

[23] X. Zhao, W. Deng, and Y. Shi, "Feature selection with attributes clustering by maximal information coefficient," *Procedia Computer Science*, vol. 17, pp. 70–79, 2013.

[24] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-supervised network traffic classification," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 369–370.

[25] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso *et al.*, "Gt: picking up the truth from the ground for internet traffic," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 5, pp. 12–18, 2009.

[26] "http://mawi.wide.ad.jp/mawi/."

[27] Z. J. Dong Yuning and J. Jiong, "Novel feature selection and classification of internet video traffic based on a hierarchical scheme," *Computer Networks*, vol. 119, pp. 102–111, 2017.