Robust Network Flow Classification against Malicious Feature Manipulation

Yupeng Li
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
yupeng.li@utoronto.ca

Ben Liang
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Canada
liang@ece.utoronto.ca

Ali Tizghadam
Technology Strategy and Business Transformation
TELUS Communications
Toronto, Canada
ali.tizghadam@telus.com

***Abstract*—Network flow classification is essential to proper provisioning of Quality of Service (QoS). Conventional machine-learning based flow classification methods assume reliable knowledge of the flow features. However, in practice, malicious flow generators can manipulate the flow features to increase the likelihood of certain learning outcomes, e.g., in terms of the QoS requirement label. Training a classifier that is robust to such feature manipulation is imperative. In this work, we present a study on robust flow classification against malicious feature manipulation. We leverage a detailed system model to capture the relation between the classifier and malicious flow generators and propose a Stackelberg-game based solution framework to train a robust classifier. We conduct extensive experimentation using real-world traces. For flows with manipulated features, the Stackelberg classifier trained by our solution framework significantly outperforms a non-robust classifier that is oblivious to manipulation, achieving accuracy close to that of the non-robust classifier on unmanipulated flows. Furthermore, the Stackelberg classifier on manipulated test flows is no worse than the non-robust classifier on unmanipulated flows.**

I. Introduction

Network flow classification is crucial for network resource management, especially to improve Quality of Service (QoS) [1]. Classical port-based or payload-based approaches are severely ineffective, especially for encrypted traffic [2]–[4]. A series of recent works have proposed methods that employ machine learning techniques and shown promising results [2]–[10]. These methods typically use only the observable flow features, such as the minimum, mean, maximum, and standard deviation of packet lengths and packet inter-arrival times.

A common assumption made in these methods is reliable knowledge of the flow feature values. However, this assumption may not hold, especially when *malicious* flow generators exist. Such generators have a vested interest in the classification outcome. They manipulate the features of their flows to game the classifier for the purpose of increasing the likelihood of outcomes favorable to themselves. For example, a malicious flow generator can change the packet inter-arrival times and the packet size in a flow in an attempt to disguise itself to evade being blocked [11], or to be prioritized for more network bandwidth so that the flow is completed faster. Though feature manipulation can incur a cost [12], [13], the overall benefit to a malicious generator may be positive.

Such malicious behavior can render conventional statistics-based methods ineffective. Specifically, malicious flow generators may be able to manipulate the flow features to best respond to the classification model committed by the classifier. Therefore, a flow from a malicious generator can be misclassified, e.g., in terms of the QoS requirement level. For example, as explained in Sec. V, our experiments with real-world traces suggest that a classifier that is oblivious to such malicious behavior can have a classification accuracy down to below 40%.

Thus, a classifier that is robust to feature manipulation is imperative. To the best of our knowledge, none of the existing flow classification methodologies was designed against malicious feature manipulation. In this work, we study the open problem of *robust flow classification*. The task is to classify flows into multiple classes corresponding to different QoS levels, aiming to map each flow to its true required QoS level. For simplicity in this initial investigation, we consider the linear classification model, which can be executed efficiently and is commonly used for flow classification in practice [14]. Our goal is to obtain a flow classifier that is robust to malicious manipulation.

To obtain such a robust flow classifier is challenging. First, the feature manipulation of a malicious flow generator is given as a best response to the classification model. Thus, the presented features might be a function of the classification model itself, which complicates the design space. Second, the features are manipulated after the classifier commits to a model. Such *ex ante* model can hardly best respond to any malicious manipulation. Third, no training data with manipulated features are available for training the classifier.

In this work, we present a system model to capture traffic flows, classifiers, and feature manipulation. We propose a solution framework based on the Stackelberg game to train a robust network flow classifier (see Fig. 1), which we term the *Stackelberg classifier*. The framework supposes that the flow features can be manipulated during model training. The classifier, after solving a carefully formulated multi-player Stackelberg game, commits to a classification model

This work has been funded by grants from TELUS and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

corresponding to the solution, i.e., a Stackelberg equilibrium, to the game. We conduct extensive experiments using real-world flow traces [15]. Our experimental results show that, in the case of manipulated flows, the proposed Stackelberg classifier significantly outperforms a non-robust classifier that is oblivious to malicious flow feature manipulation in terms of the classification accuracy. Further, the Stackelberg classifier on manipulated test flows can achieve a classification accuracy (up to 0.9) that is higher than that of the non-robust classifier on unmanipulated test flows. When classifying flows with unmanipulated features, the performance gap between the Stackelberg classifier and the non-robust classifier is small.

The rest of the paper is organized as follows. We first discuss important related works in Sec. II. Then, we present the system model and the Stackelberg-game based solution framework in Secs. III and IV respectively, followed by performance evaluations in Sec. V. Finally, we give concluding remarks in Sec. VI.

II. Related Works

Statistics-based flow classification approaches take advantage of machine learning techniques such as supervised [3], [5], [16]–[18], unsupervised [8], [19], [20], and semi-supervised learning [2], [10], [21]. However, these methods are oblivious to malicious flow feature manipulation, rendering them unsuitable for robust flow classification.

A few recent works study robust flow classification without feature manipulation [3], [20], [22], [23]. To identify flows from a mixture of known and unknown applications, Erman et al. [20] integrated a set of supervised training data with unsupervised learning. Zhang et al. [3] addressed the problem of zero-day applications in traffic classification, and proposed a robust binary classifier that can identify flows of zero-day applications and accurately discriminate predefined application classes. Wang et al. [22] proposed to combine flow clustering based on application signatures. Wang et al. [23] proposed a flow classifier that is robust to mislabelled training samples, which incorporates noise elimination and suspected noise reweighing. These works are different from ours in that none of them takes into account malicious feature manipulation.

Recently, an increasing number of works investigate generic machine learning models that are trained or tested on data controlled by adversarial agents [24]. Some of them concentrate on classification [25]–[27]. They consider only one agent that can manipulate the data features and their approaches are applicable to binary classification. Different from these works, our solution framework is based on a specifically formulated system model with multiple flow generators that captures practical issues in multi-label traffic flow classification.

III. System Model

A. Traffic Flows

We consider traffic flows each with M predefined flow features, denoted by $\mathbf{x}_i \in \mathbb{R}^M$. For instance, we can take standard features such as packet inter-arrival times and packet lengths as two of the flow features. Each flow has a truthful level of QoS requirement $k \in \mathcal{K} = \{1, 2, \cdots, K\}$ [17], [28]. Without loss of generality, we assume that the higher k is, the weaker is the QoS requirement, for example, less delay sensitive. Let $\mathbf{y}_i \in \mathbb{R}^K$ be a one-hot vector denoting flow i's true level of QoS requirement. We denote the k-th element of $\mathbf{y}_i$ as $\mathbf{y}_i^{(k)}$. Thus, if the true level is k^*, then $\mathbf{y}_i^{(k^*)} = 1$ and $\mathbf{y}_i^{(k)} = 0, \forall k \neq k^*$. As an example, in our experiments shown in Sec. V, the true level of QoS requirement of each flow is assigned according to the application it belongs to. Note that the flow features we choose do not include the application information as such information is difficult to acquire in most scenarios. Thus, each flow is associated with a tuple $(\mathbf{x}_i, \mathbf{y}_i)$.

The service provider allocates transmission resource to each flow according to its prediction of the flow's label, and each flow generator receives its utility given the service provider's resource allocation. A flow generator corresponds to an application, a user, or some other entity who is concerned about the QoS level of the flow. To represent the generator's utility, we consider a general loss function $L_g(\hat{\mathbf{y}}_i, \mathbf{y}_i)$ where $\hat{\mathbf{y}}_i$ is the predicted *score* of flow i. In the tradition of machine learning, a classification model first computes the score vector $\hat{\mathbf{y}}_i$, based on which a specific label is finally assigned. In this work, for simplicity of illustration, we assume $L_g(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \mathbf{b}^T\hat{\mathbf{y}}_i$, where $\mathbf{b} \in \mathbb{R}^K$ is a weight vector characterizing how an assigned label would affect the performance of a flow. One can employ other appropriate loss models here.

B. Classifier

We consider a linear classifier with a decision function $\hat{\mathbf{y}}_i = \mathbf{A}^T\mathbf{x}_i$, where matrix $\mathbf{A} \in \mathbb{R}^{M\times K}$. The scores are then mapped to the predicted QoS level by model $h_c(\mathbf{x}_i) = \arg\max_k \hat{\mathbf{y}}_i^{(k)}$. It is beneficial to use a linear classifier for flow classification, especially for classifying flows on the fly, as it can be efficiently executed. Our evaluation in Sec. V demonstrates that a linear classifier can achieve sufficient classification accuracy.

Denote by $L_c(\hat{\mathbf{y}}_i, \mathbf{y}_i)$ the loss function of the classifier with respect to flow i. The loss function in this work is general, but in our experiments, we adopt the commonly used *Categorical Cross-Entropy Loss* as an example [29]. We refer readers to Sec. V for more details. We use the Frobenius norm of $\mathbf{A}$, $R_c(\mathbf{A}) = \sum_i \sum_j A_{i,j}^2$, as a regularizer to reduce overfitting. Therefore, the classifier's cost function is $\mathcal{C}_c(\mathbf{A}, \mathbf{y}_i, \hat{\mathbf{y}}_i) = L_c(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \gamma_c R_c(\mathbf{A})$, where $\gamma_c \geq 0$ controls the trade-off of the regularizer against the loss. Since $\hat{\mathbf{y}}_i(\mathbf{x}_i)$ is a function of $\mathbf{x}_i$, we can rewrite $\mathcal{C}_c(\mathbf{A}, \mathbf{y}_i, \hat{\mathbf{y}}_i)$ as $\mathcal{C}_c(\mathbf{A}, \mathbf{y}_i, \mathbf{x}_i)$.

C. Feature Manipulation

Flow generators manipulate flow features to game the classifier in order to obtain better performance. In a high-speed network, it is generally expensive to build hardware for packet flow manipulation at line rate [12], [13]. On the other hand, software-based packet flow manipulation requires frequent interaction with the memory (e.g., read

TABLE I: Summary of Key Notation.

Notations	Meaning
M	# of selected features
K	# of classes
$\mathbf{x}_i$	Features of flow i
$\hat{\mathbf{x}}_i$	Manipulated features of flow i
$\mathbf{y}_i$	True label of flow i
$\hat{\mathbf{y}}_i$	Predicted score of flow i
$\mathbf{A}$	Parameters of the linear classifier
L_g	Loss of flow generator
$\mathbf{b}$	Weighting vector of the loss of flow generator
L_c	Loss of classifier
C_m	Manipulation cost of flow generator
R_c	Regularizer of classifier
$\mathcal{C}_g$	Cost of flow generator
$\mathcal{C}_c$	Cost of classifier

and write operations). Furthermore, some manipulation such as changing header fields requires more complex operation. Therefore, software-based manipulation at line rate is non-trivial and often reduces the performance of the flow. Thus, we assume that a manipulation cost $C_m(\hat{\mathbf{x}}_i, \mathbf{x}_i)$ is incurred when the feature is manipulated from $\mathbf{x}_i$ to $\hat{\mathbf{x}}_i$. In this work, we take $C_m(\hat{\mathbf{x}}_i, \mathbf{x}_i) = \|\mathbf{a} \circ (\hat{\mathbf{x}}_i - \mathbf{x}_i)\|^2$, where $\mathbf{a} = [a_1, a_2, \cdots, a_M]^T$ is a vector representing the weights of manipulating each feature.[1] Then, the cost function of a malicious generator corresponding to flow i is $\mathcal{C}_g(\mathbf{A}, \mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i) = L_g(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \gamma_g C_m(\hat{\mathbf{x}}_i, \mathbf{x}_i)$, where $\gamma_g \geq 0$ controls the trade-off of the manipulation cost against the flow's performance loss.

Key notation in this paper are summarized in Table I.

IV. Stackelberg-Game Solution Framework

Our goal is to train a classifier that is robust to feature manipulation using a set of traffic flows (with original features) as training data. Suppose the set $\mathcal{N}$ of n flows are the flows used to train our classifier, which we call the *training flows*. Our approach is to train a classifier supposing that the features of the training flows can be manipulated, at some manipulation cost defined in Sec. III, in response to the classification model. This can be modeled as an $(n+1)$*-player Stackelberg game* played by a leader and n followers. It consists of two stages [30]. In the first stage, the classifier (as a leader) decides the model parameters $\mathbf{A}$. In the second stage, each of the n flow generators (as a follower) manipulates the original features $\mathbf{x}_i$ to features $\hat{\mathbf{x}}_i, \forall 1 \leq i \leq n$, to best respond to the classification model parameterized by $\mathbf{A}$. In this game, the strategies of the classifier and flow generators are respectively the model parameters committed and manipulated features chosen. Thus, the strategy profile of the game is $(\mathbf{A}, \{\hat{\mathbf{x}}_i\}_{i\in\mathcal{N}})$. The cost functions of the classifier and flow generators in this game are the same as those in Sec. III.

To obtain model parameters that leads to an optimal cost for the classifier, we need to find a *Stackelberg equilibrium* (SE). An SE is defined as a subgame perfect Nash equilibrium or equilibria (SPNE) of the game, i.e., a strategy profile that serves best each player, given the strategies of other players, and that entails every player in a Nash equilibrium in every subgame.[2] Note that every finite extensive-form game has at least one SPNE [31]. Obviously, our Stackelberg game can be represented as a finite extensive-form game, and thus, it has an SE. In equilibrium, the strategy profile is a solution of the following problem:

$$\min_{\mathbf{A}\in\mathbb{R}^{M\times K}} \sum_{i\in\mathcal{N}} \mathcal{C}_c(\mathbf{A}, \mathbf{y}_i, \hat{\mathbf{x}}_i) \tag{1}$$

$$\text{s.t.} \quad \hat{\mathbf{x}}_i \in \arg\min_{\mathbf{x}'_i\in\mathbb{R}^M} \mathcal{C}_g(\mathbf{A}, \mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i), \forall i \in \mathcal{N} \tag{2}$$

Each constraint above corresponds to a flow in the training data and states that the manipulated features $\hat{\mathbf{x}}_i$ minimizes the cost function $\mathcal{C}_g(\mathbf{A}, \mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i)$ of the i-th flow generator given model parameters $\mathbf{A}$ are fixed. We observe that, for given (and fixed) model parameters $\mathbf{A}$, the constraints are independent of each other. For fixed $\mathbf{A}$, a manipulated feature $\mathbf{x}_i$ of the i-th flow best responds to $\mathbf{A}$ by minimizing its own cost

$$\min_{\mathbf{x}'_i\in\mathbb{R}^M} \mathcal{C}_g(\mathbf{A}, \mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i). \tag{3}$$

That is, $\min_{\mathbf{x}'_i\in\mathbb{R}^M} L_g(\hat{\mathbf{y}}_i(\mathbf{x}'_i), \mathbf{y}_i) + \gamma_g C_m(\mathbf{x}'_i, \mathbf{x}_i)$, which is convex in $\mathbf{x}'_i$. By applying the Karush–Kuhn–Tucker conditions, we have

$$\nabla_A \mathcal{C}_g(\mathbf{A}, \mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{x}}_i, \hat{\mathbf{y}}_i) = 0. \tag{4}$$

That is,

$$\mathbf{A}\mathbf{b} + \frac{\gamma_g}{2}\mathbf{a} \circ (\hat{\mathbf{x}}_i - \mathbf{x}_i) = 0. \tag{5}$$

Define $\mathbf{a}' = [\frac{1}{a_1}, \frac{1}{a_2}, \cdots, \frac{1}{a_M}]^T$. Thus, $\hat{\mathbf{x}}_i$ is uniquely defined as

$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{2\gamma_g}\mathbf{a}' \circ (\mathbf{A}\mathbf{b}), \text{ for all } i \in \mathcal{N}. \tag{6}$$

By substituting $\hat{\mathbf{x}}_i$'s into the above problem, we have

$$\min_{\mathbf{A}\in\mathbb{R}^{M\times K}} \sum_{i\in\mathcal{N}} L_c(\hat{\mathbf{y}}_i(\mathbf{x}_i) - \frac{1}{2\gamma_g}\mathbf{A}^T(\mathbf{a}'\circ(\mathbf{A}\mathbf{b})), \mathbf{y}_i) + \gamma_c R_c(\mathbf{A}). \tag{7}$$

The convexity of this problem depends on the loss function L_c of the classifier chosen. We can solve it using efficient algorithms such as SLSQP [32] and FastCubic [33]. We denote the optimal solution of problem (7) as $\mathbf{A}^*$. When there are multiple optimal solutions, we can tie break arbitrarily. After that the classifier commits to a classifier parameterized by $\mathbf{A}^*$.

Fig. 1 illustrates our solution framework to train the proposed Stackelberg classifier. We term the classifier trained by this solution framework the *Stackelberg classifier*.

[1]Here, $\|\cdot\|$ is the Euclidean norm and "$\circ$" denotes element-by-element multiplication.

[2]In our game, a subgame is either the Stackelberg game itself or the game in which model parameters are given and fixed and flow generators are the only players.

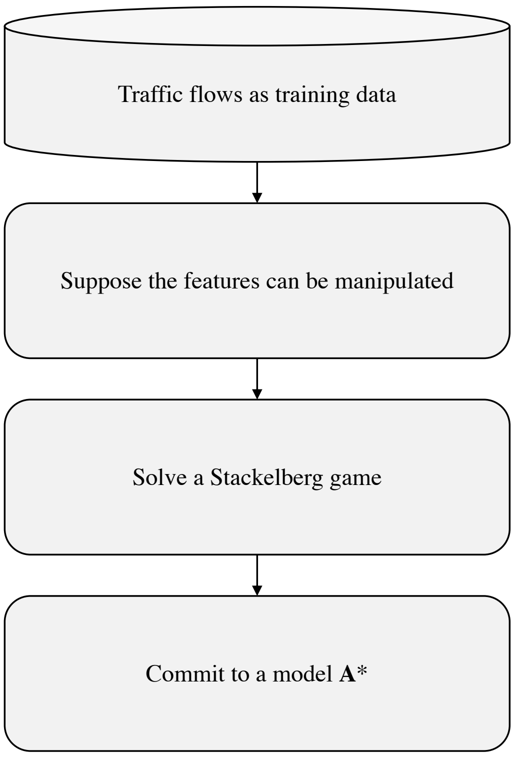
Traffic flows as training data
Suppose the features can be manipulated
Solve a Stackelberg game
Commit to a model **A***

Fig. 1: A Stackelberg-game based solution framework to train a robust classifier.

V. Experimental Evaluation

A. Methodology

1) Data Trace: We use packet trace for different periods of the 24-hour day from [15], [34]. All packets in the trace are TCP packets. One TCP connection corresponds to a flow. The data trace contains 377,526 flows in total. Each record of the trace consists of a variety of characteristic features for a flow. We consider 100 standard features including the minimum, mean, maximum, and standard deviation of client to server (and server to client) packet lengths and packet inter-arrival times, number of client to server (and server to client) packets and bytes, and the duration of the network connection. They are features numbered 3-9, 195-208, 10-30, 153-194, 210-215, 31-40 in [18]. These features are chosen since they are easier to manipulate by a flow generators in practice. Note that the chosen flow features do not contain the application type, as such information is usually difficult to acquire.

For performance evaluation, we extract from the packet trace 12 application types, which are www, mail, ftp-control, ftp-pasv, attack, p2p, database, ftp-data, multimedia, services, interactive, and games. We further assume four different QoS levels ($k = 1, 2, 3, 4$). We assign each flow a QoS level (as its true label) according to the application it belongs to, roughly based on the application's *delay* requirement, as follows: $k = 1$: multimedia, interactive, games, and ftp-control; $k = 2$: attack, www, and p2p; $k = 3$: database, ftp-data, and services; $k = 4$: mail and ftp-pasv. The distribution of the flows with different QoS levels is unbalanced. Thus we use the Synthetic Minority Oversampling Technique (SMOTE) [35] for balance, which generates new instances from existing minority classes but does not change the number of majority classes. We divide the datasets into 4:1 training and testing subsets uniformly randomly.

2) Performance Metrics and Baselines: We take the testing *error rate* as our performance metric. For comparison, we consider a non-robust classifier obtained by minimizing the total cost $\sum_{i\in\mathcal{N}} \mathcal{C}_c(\mathbf{A}, \mathbf{y}_i, \mathbf{x}_i)$ defined over the unmanipulated features $\mathbf{x}_i$ for all $i \in \mathcal{N}$. We compare the error rate of the proposed Stackelberg classifier (denoted by **Stackelberg-M**) with the following three baselines of error rates, corresponding to specific trained models and testing data (flows with unmanipulated or manipulated features):

- **Non-Robust-M**: non-robust classifier tested by manipulated features.
- **Non-Robust-U**: non-robust classifier tested by unmanipulated features.
- **Stackelberg-U**: Stackelberg classifier tested by unmanipulated features.

3) Experiment Setting: We adopt the Categorical Cross-Entropy Loss as the classifier's loss function, which is commonly used in practice and defined as $L_c(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log \frac{e^{\hat{\mathbf{y}}_i^T \mathbf{y}_i}}{\sum_k e^{\hat{\mathbf{y}}_i^{(k)}}}$ [29]. The optimizer used in our experiments is SLSQP [32]. We set $\mathbf{a}$ to an all-ones vector. We evaluate the performance of the Stackelberg classifier in different settings by varying the values of key parameters around the default setting of $(\gamma_c, \gamma_g, \mathbf{b}) = (1, 100, [8, 4, 2, 1])$. In each experiment, either Stackelberg or the non-robust classifier is trained for 30 times, each with a newly splitted training and testing subsets. All experiments are run on a machine with two Intel(R) Xeon(R) CPU E5-2650 v4 2.20GHz with 32GB memory and 1.8TB hard drive.

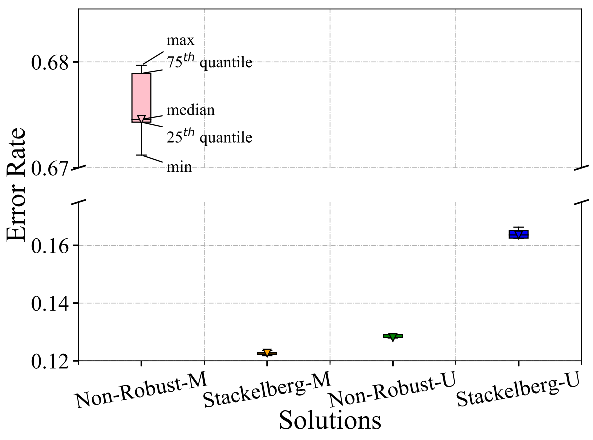

Fig. 2: Performance of the Stackelberg classifier and the baselines.

B. Performance of Stackelberg Classifier

Fig. 2 illustrates the performance of the Stackelberg classifier and the baselines. We make the following ob-

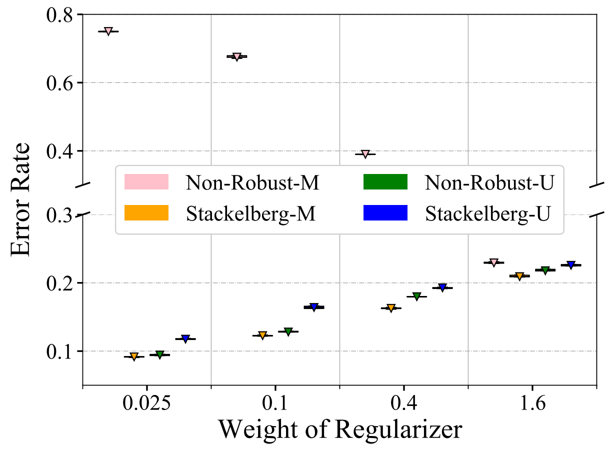

Fig. 3: Impact of weight of regularizer γ_c.

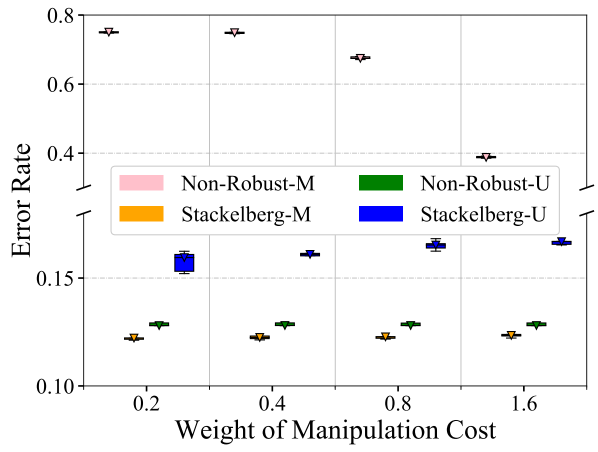

Fig. 4: Impact of weight of manipulation cost γ_g.

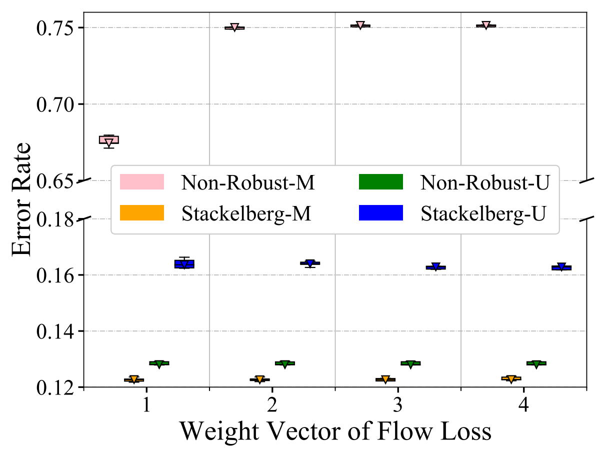

Fig. 5: Impact of weight vector of flow loss $\mathbf{b}$ set to $[8, 4, 2, 1]$ (1), $[16, 8, 4, 1]$ (2), $[32, 16, 8, 1]$ (3), $[128, 64, 32, 1]$ (4), $[256, 128, 64, 1]$ (5).

servations: 1) When the testing data are manipulated in response to the classification model, the Stackelberg classifier can decrease the error rate by up to (0.675-0.122=) 0.553, compared with the non-robust classifier. 2) The error rate achieved by the Stackelberg classifier when facing manipulated flow features (0.122) are less than the error rate achieved by the non-robust classifier tested by unmanipulated features (0.13). 3) Further, when the features of testing traffic flows are unmanipulated, the gap of the performance in classification accuracy between Stackelberg classifier and non-robust classifier is (only) (0.161-0.13=) 0.031.

Our results reveal that *one can replace the non-robust classifier with the Stackelberg classifier to classify flows generated by either malicious or normal flow generators.*

C. *Impact of Parameters on Error Rate*

We evaluate the impact of learning and systems parameters on the Stackelberg classifier's performance in terms of the error rate. In each experiment, based on the default parameter setting, we vary the parameter concerned.

1) Weight of Regularizer γ_c: We evaluate the impact of the weight of regularizer γ_c as it is varied among 0.025, 0.1, 0.4, and 1.6. Fig. 3 shows the results. When the testing data are manipulated in response to the classification model, the Stackelberg classifiers maintain superior performance in classification accuracy. When the features of testing traffic flows are unmanipulated, the gap of the performance in classification accuracy between Stackelberg and non-robust classifiers is at most 0.03 (corresponding to $\gamma_c = 0.1$). Note that the regularizer plays an important role to lessen the chance of overfitting when training a classification model. We observe that $\gamma_c = 0.025$ gives our Stackelberg classifier (and the non-robust classifier when the features are unmanipulated) the best performance. Our result sheds light on tuning the parameters in the classification model in practice.

2) Weight of Manipulation Cost γ_g: Recall that $\gamma_g \geq 0$ controls the trade-off of the manipulation cost against the loss. In this experiment, we evaluate the impact of γ_g on the performance of the proposed Stackelberg classifier and the baselines. Note that the baseline Non-Robust-U is trained and tested using the unmanipulated original features. Thus, it is not affected by γ_g. We can observe in Fig. 4 that the Stackelberg classifiers maintain superior and stable performance in classification accuracy (=0.875) when the flow features are manipulated. Further, we can find that the error rates that Non-Robust-M achieves decrease with the increase of γ_g. The reason is that higher manipulation cost leads to lower likelihood for the (malicious) flow generators to manipulate their features.

3) Weight Vector of Flow Loss $\mathbf{b}$: In this experiment, we evaluate the impact of weight vector of the loss function $\mathbf{b}$ on the performance of the algorithms. $\mathbf{b}$ is set to $[8, 4, 2, 1]$ (1), $[16, 8, 4, 1]$ (2), $[32, 16, 8, 1]$ (3), $[128, 64, 32, 1]$ (4), $[256, 128, 64, 1]$ (5). Note that the baseline Non-Robust-U is not affected by $\mathbf{b}$ as it is trained and tested using the unmanipulated original features. In Fig. 5, we can observe that, Stackelberg-M and Stackelberg-U have stable performances in the error rate respectively while Non-Robust-M does not. With the increase of the variance in the weights to different labels (i.e., QoS levels), the chance for a malicious flow generator to manipulate its flow features increases. Thus, the results show that the Stackelberg classifier is robust to the manipulation behaviors of the malicious flow generators.

VI. Conclusion

We have studied the problem of robust flow classification under malicious feature manipulation. We consider a detailed system model and propose a solution framework based on Stackelberg games. Extensive experiments on real-world flow traces show that the proposed Stackelberg classifier trained by our proposed solution framework significantly outperforms the non-robust classifier that is oblivious to malicious flow feature manipulation. When classifying flows with the original unmanipulated features, the gap of the performance between the Stackelberg classifier and the non-robust classifier is small. Therefore, one can apply the Stackelberg classifier to classify flows generated by either malicious or normal flow generators. We have also evaluated the impact of key parameters, e.g., the weight of regularizer, the weight of manipulation cost, and the weight vector of flow loss, on the classification accuracy. Our experimental evaluation sheds light on how to tune the parameters in the classification model in practice as well. For example, the weight of regularizer $\gamma_c = 0.025$ gives the Stackelberg classifier the best performance. In practice, feature manipulation can be performed adaptively in response to the latest classification model committed by the classifier. It is interesting to study robust online flow classification against such adaptive malicious feature manipulation.

References

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

[2] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[3] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Transactions on Networking (TON)*, vol. 23, no. 4, pp. 1257–1270, 2015.

[4] T. T. Nguyen, G. Armitage, P. Branch, and S. Zander, "Timely and continuous machine-learning-based classification for interactive ip traffic," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1880–1894, 2012.

[5] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity," in *Proc. ACM SIGKDD*, 2017.

[6] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 133–147, 2013.

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

[8] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.

[9] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proc. ACM SIGCOMM MineNet*, 2006.

[10] S. Rezaei and X. Liu, "How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets," *arXiv preprint arXiv:1812.09761*, 2018.

[11] M. Dusi, M. Crotti, F. Gringoli, and L. Salgarelli, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Computer Networks*, vol. 53, no. 1, pp. 81–97, 2009.

[12] S. Pontarelli, M. Bonola, and G. Bianchi, "Smashing SDN 'built-in' actions: rogrammable data plane packet manipulation in hardware," in *Proc. IEEE NetSoft*, 2017.

[13] M. Meitinger, R. Ohlendorf, T. Wild, and A. Herkersdorf, "A programmable stream processing engine for packet manipulation in network processors," in *Proc. IEEE ISVLSI*, 2007.

[14] Y. Jin, N. Duffield, J. Erman, P. Haffner, S. Sen, and Z.-L. Zhang, "A modular machine learning system for flow-level traffic classification in large networks," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, p. 4, 2012.

[15] "Nprobe: Scalable Network Monitoring Architecture," https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/nprobe/, 2019.

[16] 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, 2012.

[17] M. Lopez-Martin, B. Carro, J. Lloret, S. Egea, and A. Sanchez-Esguevillas, "Deep learning model for multimedia quality of experience prediction based on network flow packets," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 110–117, 2018.

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

[19] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, G. Wei, and L. T. Yang, "Internet traffic classification using constrained clustering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2932–2943, 2013.

[20] 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.

[21] X. Li, F. Qi, D. Xu, and X. Qiu, "An internet traffic classification method based on semi-supervised support vector machine," in *Proc. IEEE ICC*, 2011.

[22] Y. Wang, Y. Xiang, and S.-Z. Yu, "An automatic application signature construction system for unknown traffic," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 13, pp. 1927–1944, 2010.

[23] B. Wang, J. Zhang, Z. Zhang, W. Luo, and D. Xia, "Robust traffic classification with mislabelled training samples," in *Proc. IEEE ICPADS*, 2015.

[24] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proc. ACM AISEC*, 2011.

[25] J. Dong, A. Roth, Z. Schutzman, B. Waggoner, and Z. S. Wu, "Strategic classification from revealed preferences," in *Proc. ACM EC*, 2018.

[26] M. Brückner, C. Kanzow, and T. Scheffer, "Static prediction games for adversarial learning problems," *Journal of Machine Learning Research*, vol. 13, no. Sep, pp. 2617–2654, 2012.

[27] M. Hardt, N. Megiddo, C. Papadimitriou, and M. Wootters, "Strategic classification," in *Proc. ACM ITCS*, 2016.

[28] R. L. Gomes and E. R. M. Madeira, "A traffic classification agent for virtual networks based on QoS classes," *IEEE Latin America Transactions*, vol. 10, no. 3, pp. 1734–1741, 2012.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[30] H. Von Stackelberg, *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[31] M. J. Osborne *et al.*, *An introduction to game theory*. Oxford university press New York, 2004, vol. 3, no. 3.

[32] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Science & Business Media, 2006.

[33] N. Agarwal, Z. Allen-Zhu, B. Bullins, E. Hazan, and T. Ma, "Finding approximate local minima faster than gradient descent," in *Proc. ACM STOC*, 2017.

[34] A. Moore, D. Zuev, and M. Crogan, "Discriminators for use in flow-based classification," https://www.cl.cam.ac.uk/~awm22/publication/moore2005discriminators.pdf, Tech. Rep. RR-05-13, 2013.

[35] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.