"*One Model Fits All Nodes*": Neuron Activation Pattern Analysis-Based Attack Traffic Detection Framework for P2P Networks

Songsong Xu, Chuanpu Fu, *Graduate Student Member, IEEE*, Qi Li, *Senior Member, IEEE*, and Ke Xu, *Fellow, IEEE*

***Abstract*— Machine learning (ML) based network attack traffic detection is an emerging security paradigm, which is capable of capturing various advanced network attacks according to the features of traffic. When leveraging such promising security application to protect P2P services, particularly distributed cryptocurrency systems, one detection model should be deployed on many nodes to handle various unseen traffic patterns generated by nodes around the world. However, unseen yet benign traffic patterns are commonly classified as attack traffic, and thus trigger massive false-positive (FP) alarms. Unfortunately, the common practice of retraining models to reduce FPs is not salable for large-scale P2P networks, which incurs prohibitive labor efforts of collecting traffic on each node individually. To effectively deploy ML based attack traffic detection systems to protect distributed networks, we present *tNeuron* that automatically identifies FPs triggered by unseen traffic via neuron activation pattern analysis, such that it significantly improves the performance on various nodes. Specifically, we construct a shadow model with Transformer encoders to extract the knowledge of traffic patterns. Afterward, we train a model that learns how to classify FPs among alarms raised by ML models according to neuron activation patterns of the shadow model. Our experiments on real Ethereum nodes show that *tNeuron* can reduce 83.40% FP for seven state-of-the-art ML based attack detection systems, when detecting 15 kinds of P2P network attacks, thereby significantly improving detection accuracy in nine different metrics. In addition, *tNeuron* is robust against various adversarial examples constructed by existing evasion attacks. Besides, it achieves real-time detection and is capable of handling massive FPs generated by many nodes in large-scale distributed networks.**

***Index Terms*— Network security, machine learning (ML), P2P network, intrusion detection.**

I. Introduction

MACHINE learning (ML) based attack traffic detection is an emerging and promising security paradigm [24], [25], [54], [63], [67], which aims to capture network attacks according to the features of traffic, for instance, duration and transferred bytes of a flow. It can outperform traditional fixed-rule based methods [50], [76], [78], [79] by identifying various sophisticated and zero-day network attacks [10], [21], [29], and thus effectively protect many critical Internet services [59], [63], [67], [81].

Currently, ML based traffic detection systems are commonly deployed at gateways of networks [5], [8], [80], where traffic datasets containing all traffic patterns from/to protected Internet services are collected for ML training and validation. In P2P networks, where such centralized gateways are absent, deploying ML models to safeguard decentralized cryptocurrency applications—comprising many globally located nodes, such as Ethereum [19] and Bitcoin [9]—presents a trade-off between cost and effectiveness. On one hand, the ML pipeline is labor-intensive [3], [35], making it impractical to label diverse traffic patterns from distributed node and to train one model per node. On the other hand, deploying one model across all distributed nodes leads to significant accuracy penalties due to numerous false-positive alarms (FPs) triggered by benign yet previously unseen traffic [1], [16], [45]. Our empirical studies show that, one well-trained model, which is deployed to an Ethereum node other than where it is trained, generates over 13.0K FPs per hour, which are triggered by benign traffic deviating from training samples.

To enable distributed deployment in P2P networks, we aim to minimize FPs, which ensures that one model functions well across all nodes, thereby safeguarding each decentralized P2P node from various attacks [7], [33], [62], [69], [71]. Note that, existing FP reduction workflows incur huge labor consumption [8], [15], [16], [68], i.e., experts should manually identify FPs, before they can retrain the model using identified FPs [30], [45]. Unfortunately, due to the scale of P2P nodes [7], [14], it is impossible to manually identify even a minority of FPs, which leads to high latency [3], prohibitive monetary consumption [45], and the fatigue of alarm issue that directly hinders the security of P2P system [1]. To the best of our knowledge, automatic FP detection for P2P networks without requiring any labor is still unavailable, which is the key challenge for training one attack detection model that fits all nodes for throttling malicious traffic targeting critical P2P cryptocurrency applications [2], [33], [62], [69], [71].

The recent advances in brain science motivate this study. Particularly, existing studies demonstrate that the fatigue status of human brain, which leads to poor decisions, reflects on

Received 24 January 2024; revised 19 September 2024 and 8 January 2025; accepted 9 February 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor G. Fanti. This work was supported in part by China National Funds for Distinguished Young Scientists under Grant 62425201, in part by the National Natural Science Foundation of China under Grant 62132011 and Grant U22B2031, and in part by the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant 62221003. *(Songsong Xu and Chuanpu Fu contributed equally to this work.) (Corresponding author: Ke Xu.)*

Songsong Xu, Chuanpu Fu, and Ke Xu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: haizhitiantang1@163.com; fuchuanpu@gmail.com; xuke@tsinghua.edu.cn).

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TON.2025.3546735

chemical status of brain neuron cells [32], [56]. Similarly, we postulate that status of underfitting unseen traffic samples, which results in false-positive alarms, can be likewise recognized by analyzing neuron activation patterns of deep learning models. Thus, we aim to build a model to learn if a traffic detection model is producing an FP alarm, given the empirical observation that, the neurons activated by unseen P2P traffic that triggers FPs are significantly different from that are properly activated by well-learned samples when producing correct results.

In this paper, we propose *tNeuron*, a realtime system that reduces FPs raised by ML based attack traffic detection models, when deploying these models to any nodes other than where they are trained. To effectively perform neural activation patterns analysis, we construct a shadow model that extracts the knowledge of traffic patterns. Specifically, the model is built upon Transformer encoder [13] which is trained to predict masked fields in packets [44]. These model provide rich neurons for analysis, as existing models may contain limited numbers of neurons [54], [81] or employs traditional learning models. After that, we train a model to learn the neuron activation patterns of the Transformer encoders in an unsupervised manner, such that the model can classify alarms into TPs and true positives (TPs) by checking if neurons are properly activated. Please note that the training process of *tNeuron* does not require a centralized view of the P2P network. Instead, it only requires traffic data from a single node, enabling decentralized deployment of the trained model across all distributed networks.

We conducted experiments using traffic collected from real Ethereum nodes. The experimental results demonstrate that, *tNeuron* can reduce over 90% FPs raised by seven state-of-the-art ML based traffic detection methods [5], [11], [24], [26], [35], [50], [80] under various attacks. These methods cover a broad spectrum of ML based attack detecting, including flow based [80], packet based [35], host based [24] detection that employs various supervised and unsupervised ML models. By reducing FP alarms, *tNeuron* enhances the performance in distributed network of these systems by improving nine accuracy metrics, which suggests *tNeuron* enables one traffic detection model to fit the nodes generating unseen traffic without incurring any additional labor costs. Additionally, the experiments show that *tNeuron* has no negative effects on the robustness of ML models [24], [26], so that attacks cannot launch evasion attacks when applying *tNeuron* for offering all-node protection for P2P applications. Furthermore, *tNeuron* can reduce the FPs in real-time with a low latency of 0.0164s, while achieving 12.82K alarm processing throughput per second.

In general, the contributions of this paper are four-fold:

- We present *tNeuron*, the first system that reduces the accuracy penalty caused by deploying ML based traffic detection in P2P networks.
- We develop a knowledge extraction method using Transformer based traffic classifier that provides informative neuron activation patterns for analysis.
- We design a model that predicts FPs according to the neuron activation patterns of the Transformer model.
- We prototype *tNeuron* and use extensive experiments with various real-world traffic datasets to validate its effectiveness and efficiency.

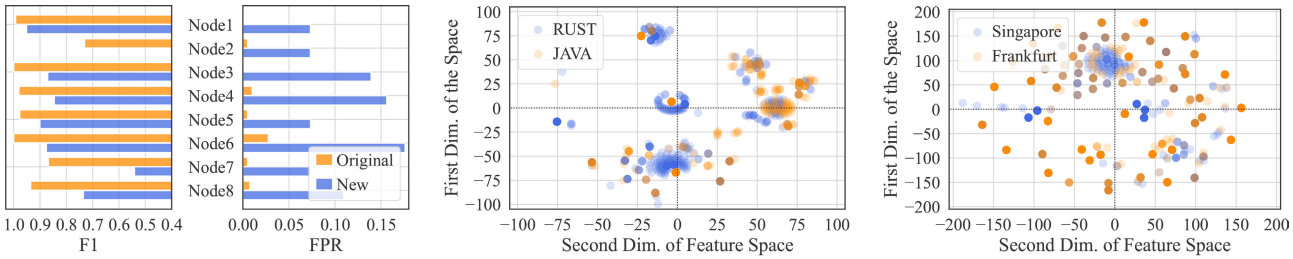
The rest of this paper will be organized as follows. In section II, we will formulate the problem. Section III dives into the design details of *tNeuron*. In section IV, we experimentally evaluate the performance of *tNeuron*. Section V reviews the related studies and Section VI concludes this paper.

II. Problem Statement and Design Goals

tNeuron is designed for protecting peer-to-peer (P2P) distributed networks that consist of numerous nodes located around the world [9], [19]. Specifically, we assume that a cryptocurrency company owns many Ethereum nodes [33] with various configurations, which are attractive targets for various attacks [2], [33], [57], [62], [71]. Meanwhile, the operators have already collected traffic from one of these nodes hosting P2P services. After that, the traffic dataset is used to train an ML based traffic detection model [5], [24], [25], [63], [68], which could be either supervised [35] or unsupervised [54]. Unlike traditional use cases where the model is deployed at a single point [50], [80], e.g., gateways [76] that can observe all traffic patterns, The operators should share the model across the nodes to effectively protect all the nodes owned by the company. In general, *tNeuron* aims to enable sharing one model across many nodes owned by the company, which may also incentivize the sharing of models among nodes with different ownership.

Note that, due to the decentralized nature of P2P networks, it is impossible to derive a centralized view to collect all traffic from a P2P network. Moreover, it is also impossible to collect traffic from each node and train the model for each node individually due to the complexity of the ML pipeline and prohibitive labor costs [3], [66] of collecting traffic datasets for training, tuning, and validating ML models. However, we observe that deploying a single model across all nodes results in significant accuracy decrease. Specifically, we select eight Ethereum nodes maintained by us, which vary in location, software, and hardware configurations (details provided in Table II). Subsequently, we train a random forest based flow-level detection model [80] using traffic collected from one Ethereum node to detect OpenSSH attacks [25]. The results are illustrated in Figure 1.

We observe that the model achieves high accuracy when tested on the original node where it is trained (*Original* in Figure 1(a)). However, when deploying the trained model to the seven nodes other than where the model is trained (denoted as *New*), we observe 7.69% - 56.88% F1 decrease averaged across the nodes, due to frequently triggered FP alarms, i.e., the associated FPR increases by 6.35% - 15.31%. This is because the configuration and workload of each node are different even if they host the same P2P application [2], [9], [19], [62]. Moreover, different nodes generate similar traffic patterns which can trigger massive FPs. As shown in Figure 1(b), nodes with Rust and Java Ethereum implementations execute different consensuses implementations, resulting in different



(a) Accuracy comparison on various nodes. (b) Traffic features by different implementations. (c) Traffic features by different locations.

Fig. 1. Drifted traffic pattern decreases accuracy by triggering massive FPs.

traffic patterns. Furthermore, the geological locations of nodes also significantly affect the traffic patterns, as the nodes in Singapore and Frankfurt compared in Figure 1(c). Thus, existing traffic detection systems tend to classify all these unseen yet benign traffic generated by other nodes as malicious traffic [55], [74], leading to serious issues of FP [1].

tNeuron aims to automatically detect and reduce these FPs, which enables one model to fit all P2P nodes. Different from *tNeuron*, all existing studies focus on how to utilize manually identified FPs [8], [15], [16], [68], whereas it is impossible to manually identify even a small minority of FPs generated by numerous P2P nodes. Specifically, our empirical studies observe that an existing traffic detection system can raise 13.0K alarms per hour. Therefore, automatic FP reduction is the only solution for deploying ML based traffic detection in distributed networks. In summary, we aim to design *tNeuron* as a framework that effectively deploys ML based traffic detection in distributed networks and enhances model performance by reducing FP alarms automatically. Particularly, *tNeuron* should achieve the following design goals:

- **Generic FP Detection:** *tNeuron* should detect FP raised by many different malicious traffic detection methods that employ both supervised and unsupervised ML models to analyze various traffic features [24], [54], [80].
- **Accurate FP Detection:** *tNeuron* should not misclassify true positive (TP) alarms as false positive ones which allows attackers to evade the detection, when *tNeuron* is enabled for reducing false alarms.
- **Realtime FP Detection:** *tNeuron* should achieve real-time detection for the FPs raised by massive nodes in large-scale distributed networks.
- **Robust FP Detection:** *tNeuron* should preserve the robustness of the original detection model [24], [26], i.e., adversarial examples cannot evade detection.
- **Zero Labor Cost:** *tNeuron* cannot require any human experts to identify any FPs from massive alarms generated by various nodes or access any identifies FPs.

After automatic FP identification, we can conduct model fine-tuning by utilizing the identified FPs, according to existing studies on lifelong ML paradigms [15], [16].

III. Design of *tNeuron*

A. Motivation

The motivation behind this work is the recent observations on the status of brain cells. Specifically, the chemical substances released by neurons indeed reflect the mental status, particularly, if a fatigue leads to poor decisions. Based on the similarity of brain cells and neuron modes in deep learning models, we aim to train a model to predict if a traffic detection model is generating FPs according to its neuron activation patterns [23], [48]. Specifically, abnormal neuron activation patterns indicate that the input traffic sample is an unseen pattern generated by other nodes, and thus it is not well-learned by DNN and the generated alarm may be a false one. Otherwise, the normal activation pattern indicates the input sample appears in the training dataset which is well-learned. We can thus believe the output is a true alarm.

Detecting FPs by analyzing neuron activation patterns is challenging, primarily because most traffic detection models are either non-DNN models or DNNs with a limited number of neurons [26], [54], such that only limited neuron activation information is available. To address this issue, we develop a shadow model that extracts knowledge from traffic patterns, thereby providing rich information for neuron activation pattern analysis. Specifically, we construct a Transformer encoders [13] based model which contains many neurons, to predict the masked fields [44] based on the low-level traffic features (e.g., the sizes of packets), making it easy to obtain labels without any effort. The Transformer model has high sequence learning ability so that its neutron activation patterns offer rich information. In addition, we use an unsupervised learning approach to learn the activation patterns, which does not require labor to construct labeled datasets.

In summary, *tNeuron* achieves all the design goals of FP detection (see Section II). First, *tNeuron* builds a Transformer classifier as shadow model which is independent to the original detection model, so that *tNeuron* is applicable to many detection methods [5], [24], [54], i.e., *generic FP detection*. Meanwhile, the Transformer model contains many neurons for activation pattern analysis, enabling *accurate and robust FP detection*. In addition, we leverage a light-weight model for meta-learning to realize *realtime FP detection*. Besides, the mask training of Transformer does not require labor to construct datasets, while the activation pattern model is unsupervised, and *tNeuron* thus *does not incur labor costs*.

B. Overview of *tNeuron*

Figure 2 illustrates the architecture of *tNeuron*, which is a framework to enhance the performance of traffic detection system by detecting FP alarms generated, when ML models

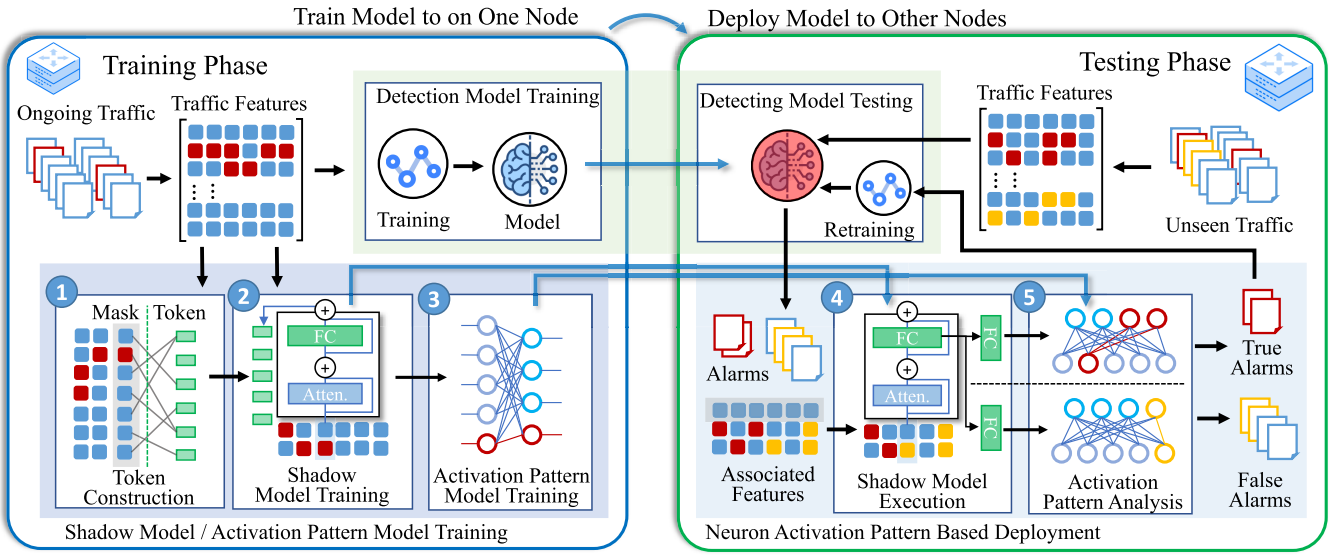


Fig. 2. The overview of *tNeuron*. At training phase, *tNeuron* constructs Transformer based surrogate model which provides rich neurons for the training of activation patterns models. At testing phase, *tNeuron* execute surrogate model for traffic triggering each alarm, and apply neuron activation model to predict if the alarm is a false one.

are deployed to P2P nodes other than where they are trained. This enables attack traffic detection for the entire P2P network, which includes a variety of different network environments.

After a model is trained by using the traffic collected from one node that hosts P2P applications (e.g., Ethereum [19] used as an instance in our experiments). *tNeuron* jointly trains two models, i.e., a Transformer based shadow model and a neuron activation model, which facilitate the detection model to reduce FPs when deploying the trained model to many other nodes in a P2P network [33]. Three steps are required to train the two models.

Token Construction. First, *tNeuron* constructs a dataset to train the shadow model. Specifically, *tNeuron* collects the packets in the training dataset for the detection model and organizes them into flows based on the five-tuples in the packet headers. After that, it tokenizes each packet in the flows by extracting per-packet features, e.g., packet length. We mask port numbers that indicate types of services, as these fields are to be predicted. Next, *tNeuron* applies the K-Means to cluster the source and destination port numbers for ingress and egress traffic, respectively. Afterward, it constructs labels for the flows based on the cluster centers associated with the port numbers.

Transformer Shadow Model Training. In this step, we construct a shadow model that encodes the low-level patterns of traffic and can provide rich neurons for activation analysis. In specific, *tNeuron* trains a classifier to predict the label constructed in the previous step based on the traffic features. In this way, *tNeuron* synthesizes the knowledge of the traffic patterns. Particularly, to effectively learn the token sequence, we develop a DNN model by leveraging stacked Transformer encoders, which are bidirectionally connected [13], and can accurately predict the masked fields. After training the model, all the flows in the training dataset are input to the trained Transformer model to derive the activation patterns of the last fully connected layer and construct the datasets for neuron activation pattern analysis.

Neuron Activation Model Training. The final step involves training an unsupervised model that learns the behaviors of the constructed traffic classification model, represented by the neuron activation patterns of the fully connected layer in the Transformer encoder based shadow model. Specifically, we leverage DBSCAN [25], an unsupervised model, to learn the patterns. In this way, the model can differentiate abnormal activation patterns, which indicate the raising of FP alarms.

In general, *tNeuron* derives the following three models: (i) a trained detection model; (ii) a Transformer based classification model (shadow model); and (iii) a neuron activation model. When the traffic detection model is deployed on a node other than training dataset is collected, numerous alarms will be raised, including many FP alarms due to the unseen traffic patterns generated by other nodes. In this scenario, operators can execute the shadow model and the neuron activation model to efficiently identify FP alarms and enhance the performance of the traffic detection in distributed networks.

Transformer Model Execution. Once an alarm is raised by the model from any nodes, the detection system collects all packets associated with the alarm. For each alarm, *tNeuron* tokenizes each of these packets and executes the shadow model to obtain the activation patterns of the fully connected layers. Note that, we do not care about the accuracy of the shadow model. Instead, we only use the neuron activation patterns to predict if the alarm is a TP or FP by executing the neuron activation pattern analysis model. Additionally, we do not use packet header fields as features, such as time-to-lives (TTLs), IP addresses, and TCP windows, because these features may directly indicate the type of traffic and lead to spurious correlation issues [37] (also known as data snooping issues [3]). Instead, *tNeuron* analyzes only the statistical features, such as packet sizes and arrival intervals, to avoid these issues.

Neuron Activation Model Execution. We input the activation patterns of the fully connected layers of the Transformer to the neuron activation model. The model calculates the distance between the pattern and the clustering centers derived in the training phase. An activation pattern that deviates from the patterns triggered by the samples in the training sets indicates abnormal behavior by the shadow model. This suggests low learning quality by the detection model, which was trained on the same training set. As a result, the associated alarm has a high probability of being an FP alarm. Please note that the shadow model can only provide activation patterns and cannot directly detect FPs, because it cannot analyze and compare the its activation patterns generated across multiple executions, to identify the abnormal activations.

Finally, the identified FPs will be assigned with low priorities for manual analysis, and such identified FPs can be used for model retraining to tune the model that raises the alarms [15], [16], [77].

C. Design Details of tNeuron

In this section, we formally describe the details of *tNeuron*. Algorithms 1 and 2 illustrate the workflows of the training and testing phases, respectively. We assume a traffic dataset $\{\mathcal{X}_1, y_1, y_p\}$ is collected from a P2P node, and another traffic dataset $\{\mathcal{X}_2, y_2\}$ is collected on the node other than where $\mathcal{X}_1$ is collected. Note that, each element in $\mathcal{X}_1$ denotes a packet sequence associated with a flow identified by a five-tuple,[1] i.e., $\mathcal{X}_1^i$ is a $L_i \times M$ matrix containing the M per-packet features of L_i packets in i-th flow ($i \in [1, N_1]$). In addition, binary label vector y_1 indicates benign and attack flows. Let $\mathcal{M}_D$ denotes the traffic detection models trained on $\mathcal{X}_1$, i.e., $\mathcal{M}_D = \mathsf{Train}_D(\mathsf{FeatureExtract}(\mathcal{X}_1), \mathsf{y}_1)$. *tNeuron* aims to boost the performance of $\mathcal{M}_D$ on $\{\mathcal{X}_2, y_2\}$, i.e., $\mathsf{Accuracy}(\mathcal{M}_D(\mathcal{X}_2), y_2)$.

Token Construction. First, *tNeuron* constructs a dataset $\{\mathcal{X}_T, y_T\}$ based on $\{\mathcal{X}_1, y_1, y_p\}$ to train the Transformer model. Specifically, we tokenize j-th packets in i-th flows in $\mathcal{X}_1$ by extracting three per-packet features, i.e., the arrival interval (us), IANA protocol number, and packet length in byte, for first L_{Max} packets. Here, we only consider first L_{Max} packets and place zero vectors in $\mathcal{X}_T^i$ to represent remaining locations for flow with lower than L_{Max} packets ($L_i \leq L_{\text{Max}}$). The tokenized dataset is denoted by $\mathcal{X}_T$. Afterward, we train the Transformer model to predict port number based labels. To construct the labels $y_T = [y_T^1, \ldots, y_T^{N_1}]$, we cluster the port numbers denoted by y_p^i using K-Means algorithm:

$$\mathcal{C} = \mathsf{KMeans}(y_p), \quad y_T^i = \arg\min_j \left\| C_j - y_p^i \right\|_2, \qquad (1)$$

where $\mathcal{C} = \{C_1, \ldots, C_K\}$ are the K cluster centers. In general, we train the shadow model to predict port numbers based on traffic patterns, because the port numbers offer diverse labels for prediction, which can be derived directly without significant labor effort.

Shadow Model Training. We construct a shadow model to provide rich information for neuron activation pattern analysis. Specifically, we design a sequential model that uses the features associated with a flow to predict which cluster it belongs to. The model contains stacked bidirectional Transformer encoders [13]. First, we expand the dimension of the tokens from L_{Max} to L_{Expand} by using a fully connected layer $\mathsf{FC}^0(\cdot)$. Also, we leverage the position encoding represented by vector e, from the existing study [13]. Thus, we feed $x_i^1 = \mathsf{ReLU}(\mathsf{FC}^0(\mathcal{X}_T^i)) + e$ into the first layer of

[1]Flows are identified by source / destination IP addresses and port numbers along with a number indicating L4 protocol types (e.g., TCP or UDP).

Algorithm 1 The Training Phase Workflow of *tNeuron*

Input: Dataset collected from one node: $\{\mathcal{X}_1, y_1, y_p\}$ denotes features, labels, and port numbers.
Output: Shadow model: $\mathcal{M}_{\mathsf{shadow}}$, activation model: $\mathcal{M}_{\mathsf{activation}}$, and detection model: $\mathcal{M}_D$.

// Train the attack traffic detection model using the dataset.
$\mathcal{M}_D = \mathsf{Train}_D(\mathsf{FeatureExtract}(\mathcal{X}_1), y_1)$
// Construct training datasets for the shadow model.
Function `TokenConstruction(`$\mathcal{X}_1$, y_p`)`**:**
  $\mathcal{X}_T$:= tokenlize each flow in $\mathcal{X}_1$.
  $\mathcal{C} := \mathsf{KMeans}(y_p)$
  $y_T^i := \arg\min_j \left\| \mathcal{C}_j - y_p^i \right\|_2$
  return $\mathcal{X}_T$, y_T
// Train the shadow model and obtain its normal activation patterns.
Function `ShadowModelTrain(`$\mathcal{X}_1$, y_T`)`**:**
  Define $\mathcal{M}_{\mathsf{shadow}}(x; \mathsf{FC}^0, \ldots, \mathsf{FC}^{N_L}, W^Q, W^K, W^V)$**:**
    e:= Positional encoding.
    $x^1 = \mathsf{ReLU}(\mathsf{FC}^0(x)) + e$
    for k *in* $[1, 2, \ldots, N_L]$ **do** // For each attention layer.
      for j *in* $[1, 2, \ldots, N_H]$ **do** // For each attention head.
        $Q_j = x^k W^{Q_j}, K_j = x^k W^{K_j}, V_j = x^k W^{V_j}$
        $A_{k,j} := \mathsf{Softmax}(\frac{Q_j K_j^T}{\sqrt{D_K}}) V_j$
      $o^k := \mathsf{FC}^k(x^k + A_k), \quad x^{k+1} := \mathsf{ReLU}(o^k)$
    $O := [o^1, \ldots, o^{N_L}]$
    return O
  for x_i *in* $\mathcal{X}_1$ **do**
    $O_i := \mathcal{M}_{\mathsf{shadow}}(x_i)$
    $\mathcal{L}_i := \frac{1}{L_{\text{Max}}} \left\| \mathsf{Softmax}(O_i^{N_L}) - \mathsf{OneHot}(y_T^i) \right\|_2^2$
    $\mathsf{AdamOptimizer}(\mathcal{L}_i, \mathcal{M}_{\mathsf{shadow}})$
  $\mathcal{O} := [O_i, \ldots, O_{N_1}]$
  return $\mathcal{O}, \mathcal{M}_{\mathsf{shadow}}$
// Train the activation model.
Function `ActivationModelTrain(`$\mathcal{O}$`)`**:**
  $a_j := \frac{1}{L_{\text{Max}}} \sum_{i=1}^{L_{\text{Max}}} \mathcal{O}_{j,i}^{N_L}, \quad \mathcal{A} := \{a_1, \ldots, a_{N_1}\}$
  $\mathcal{C}_a = \mathsf{DBSCAN}(\mathsf{MinMaxNorm}(\mathcal{A}); \epsilon, \mathsf{minPoint})$
  $\mathcal{M}_{\mathsf{activation}} := \mathcal{C}_a$
  return $\mathcal{M}_{\mathsf{activation}}$

Algorithm 2 The Testing Phase Workflow of *tNeuron*

Input: Traffic from another node: $\{\mathcal{X}_2\}$, detection model: $\mathcal{M}_D$, shadow model: $\mathcal{M}_{\mathsf{shadow}}$, and activation model: $\mathcal{M}_{\mathsf{activation}}$.
Output: Corrected decisions: $\hat{y}$.
// Execute the attack traffic detection model.
$y = \mathcal{M}_D(\mathcal{X}_2)$
// Check neuron activation patterns of the shadow model.
Define Function `ShadowModelExecution(`x`)`**:**
  $O = \mathcal{M}_{\mathsf{shadow}}(x)$
  $a = \frac{1}{L_{\mathrm{Max}}} \sum_{j=1}^{L_{\mathrm{Max}}} O_j^{N_L}$
  $d = \min_i(\left\|C_a^i - a\right\|_2), \quad C_a^i \in \mathcal{M}_{\mathsf{activation}}$
  return $d > \psi$
// Use the neuron activation patterns to correct detection results y.
for i *in* $[1, \ldots, N_2]$ **do**
  if $y_i = 1$ **then** // For the flows triggering alarms.
    // Check the activation patterns.
    if `ShadowModelExecution(`$\mathcal{X}_2^i$`)` **then**
      // Abnormal activation patterns indicate a false alarm.
      $y_i = 0$
  $\hat{y} := y$
  return $\hat{y}$

the Transformer encoder. Specifically, the Transformer encoder contains a multi-head attention layer, where the number of the heads is N_H:

$$Q_j = x_i^k W^{Q_j}, K_j = x_i^k W^{K_j}, V_j = x_i^k W^{V_j}, \tag{2}$$

$$\mathsf{Atten}_k(Q_j, K_j, V_j) = \mathsf{Softmax}(\frac{Q_j K_j^T}{\sqrt{D_K}}) V_j, \tag{3}$$

where, x_i^k is the input of the k-th layer ($1 \le k \le N_L$). Q_j, K_j and V_j denote the query, key, values matrices of the j-th attention head which are learnable. Moreover, a full connected layer is appended to the attention layer:

$$o_i^k = \mathsf{FC}^k(x_i^k + \mathsf{Concat}(\mathsf{Atten}_k(Q_j, K_j, V_j) | j \in [1, N_H]), \tag{4}$$

where o_i^k is the activation values of the k-th layer. Next, $\mathsf{ReLU}(o_i^k)$ is propagated to the next layer if $k < N_L$. The output layer is a Softmax layer, whose size equals to K. Finally, we use MSE as loss function, and back-propagate the loss to update all the parameters [13]:

$$\mathcal{L}_i = \frac{1}{L_{\mathrm{Max}}} \left\| \mathsf{Softmax}(o_i^{N_L}) - \mathsf{OneHot}(y_T^i) \right\|_2^2, \tag{5}$$

where $\mathsf{OneHot}(y_T^i)$ is the one-hot encoding of the label for the flow. The trained model is denoted by $\mathcal{M}_{\mathsf{shadow}}$.

Neuron Activation Model Training. In this setp, we generate benign activation pattern for the shadow model, and utilize DBSCAN to cluster the activation patterns. That is, the cluster centers denote benign neuron activations of the shadow model. Specifically, we extract the activation patterns of the fully connected layer of the last encoder:

$$a_j = \frac{1}{L_{\mathrm{Max}}} \sum_{i=1}^{L_{\mathrm{Max}}} o_{j,i}^{N_L}, \quad \mathcal{A} = \{a_1, \ldots, a_{N_1}\} \tag{6}$$

where a_j is the activation pattern of the j-th flow. We model the distribution of the activation patterns using DBSCAN:

$$\mathcal{C}_a = \mathsf{DBSCAN}(\mathsf{MinMaxNorm}(\mathcal{A}), \epsilon, \mathsf{minPoint}), \tag{7}$$

where ϵ and minPoint are the hyper-parameters of DNSCAN. Clustering centers $\mathcal{C}_a$ represent the benign activations of the shadow model, i.e., which neurons are activated when using traffic features to predict y_T (the clusters which the flows belong to). The activation patterns are normalized before conducing the clustering. We only analyze the activation patterns of the last layer because the patterns from the initial layers are too complex for a lightweight unsupervised ML model to analyze. In Section IV-F, our ablation studies reveal that analyzing the last two to five layers of neuron activation patterns can reduce 92.65% - 94.56% FPs, which is similar to analyzing only the last layer.

Overall, in the training phase, we derive three models, the traffic detection model $\mathcal{M}_D$, the Transformer shadow model $\mathcal{M}_{\mathsf{shadow}}$, and the neuron activation model $\mathcal{M}_{\mathsf{activation}} = \mathcal{C}_a$. Next, *tNeuron* deploys the detection model $\mathcal{M}_D$ to inspect the traffic from the other node denoted by $\mathcal{X}_2$. Meanwhile, it deploys the shadow model $\mathcal{M}_{\mathsf{shadow}}$ and the activation model $\mathcal{M}_{\mathsf{activation}}$ for FP reduction.

Shadow Model Execution. Once an alarm is raised by the detection model, i.e., $\mathcal{M}_D(\mathcal{X}_2^i) = 1$, $i \in [1, N_2]$. *tNeuron* tokenizes the i-th flow to execute the shadow model, i.e., calculating $\mathcal{M}_{\mathsf{activation}}(x_i^1)$. Note that, we execute the Transformer only if $L_i \in [L_{\min}, L_{\max}]$, because a too long or too short packet sequence cannot provide informative activation patterns. Afterward, we can derive the activation patterns of the fully connected layer which is denoted by a_i. Notably, we do not calculate the predicted ports, which means the port numbers are only used for training the shadow model. Thus, we discard the prediction result $\mathsf{Softmax}(o_i^{N_L})$. Note that although the port number may change during the testing phase—resulting in low accuracy for port number predictions by the shadow model—we focus solely on its activation patterns and are not concerned with its prediction accuracy. Furthermore, our experiments indicate that even with a 3.55% decrease in accuracy due to overfitting, the shadow model still enables *tNeuron* to reduce over 95.67% false alarms.

Neuron Activation Model Execution. Finally, we execute the model $\mathcal{M}_{\mathsf{activation}}$ to judge if the alarm is a true or false one. Specifically, we calculate the distance between the activation pattern to the nearest clustering center:

$$d_i = \min_j(\left\|C_a^j - a_i\right\|_2), \quad C_a^j \in \mathcal{C}_a. \tag{8}$$

Finally, if $d_i > \psi$, we believe that the activation pattern is abnormal, indicating that the neurons are processing unseen traffic patterns and the alarm may be a false alarm (FP). On the other hand, if the distance is below the threshold, we believe that the neurons are properly activated, indicating that the

TABLE I
DESCRIPTION OF HYPER-PARAMETERS

Category	Parameter	Default	Description
Token Construction	L_{Max}	2×10^4	Number of packets.
	K	10	Centers of K-Meams.
DNN Architecture	L_{Expand}	100	Scale of the first layer.
	N_H	10	Number of attention headers.
	N_L	12	Layer of encoders.
Activation Model	ϵ	10^{-3}	DNSCAN distance parameter.
	minPoint	50.0	DNSCAN aggregation.
	ψ	1.0	Threshold for judgement.

alarms could be a true alarm (TP). Note that, we only use light-weight model to analyze the activation patterns. This is because the shadow model provides rich information regarding the performance qualities of detection models, which allows us to easily differentiate false alarms without computation-intensive models. Moreover, the activation model is not simply a layer performing vector calculations like Softmax. Instead, it learns normal activation patterns generated by executing the shadow model multiple times, during the training phase. During the testing phase, it matches these normal patterns to reduce false alarms.

IV. EXPERIMENTAL EVALUATION

In this section, we prototype *tNeuron* and evaluate its performance by identifying FPs raised by seven state-of-the-art methods under various network attacks. In particular, the experiments will show that *tNeuron* is able to:

leftmargin=*

1) effectively identify FPs raised by various methods on different datasets (Section IV-B);
2) so that *tNeuron* can boost the performance when deploying ML model in various metrics (Section IV-C).
3) be robust against various ML models under different hyper-parameter settings (Section IV-D).
4) realize robustness detection under evasion attacks (Section IV-E).
5) perform effective neuron activation analysis by utilizing the shadow model (Section IV-F).
6) achieve high processing throughput with low latency (Section IV-G).

A. Experiment Setup

Datasets. To generate benign traffic, we collect traffic from eight real Ethereum nodes [19], i.e., a decentralized blockchain with smart contract functionality serving as the infrastructure of ETH cryptocurrency. We choose Ethereum nodes maintained by us, because their diverse implementations provide various deployment scenarios. These nodes are located around the world with different versions of software (i.e., execution and consensus clients), and various hardware configurations. Specifically, all these nodes work at Sepolia Ethereum testnet, as it supports all kinds of implementations. We observe the nodes generate various traffic patterns (see Section II for details) which trigger massive false alarms. Detailed configurations are shown in Table II. Note that, the first two nodes generated more false-positives, as the initial synchronization is not completed.

We utilize benign traffic from the Ethereum nodes as benign samples, during both training and testing phases. For malicious samples, we construct 15 typical P2P network attacks against our Ethereum nodes. According to attack goals, the datasets are classified into four groups:

- *DoS Attacks*: Attackers inject TCP-SYN and TCP-RST packets into P2P connections, to disrupt the connections between Ethereum nodes. In addition, we consider that an attacker opens massive connections without sending data. Moreover, we also implement recently disclosed Gethlighting attacks [34] that flood transaction messages.
- *Disturbing Attacks*: Attackers control Ethereum nodes that delay the reply to incoming connections by reducing 50% and 80% sending speeds. We also consider attackers may stop sending data to their peers by 10 and 30 seconds. These behaviors can disturb the stability of P2P protocols and increase the latency of the consensus.
- *Partition Attack*: On-path attackers, who control network infrastructures, try to partition some nodes, e.g., by dropping messages [70], delaying messages and manipulating IP prefixes [2]. Note that, for the prefix based attack [2], we only implement the scanning phases, as the scale of our network is much smaller. We also fake addresses for simulating eclipse attacks like existing studies [20].
- *Session Hijacking*: Off-path attackers attempt hijack the DNS connections [52] to inject their addresses into neighborhood tables, the prerequisites of many P2P attacks [75]. We also consider attackers may guess TCP sequence numbers to hijack the connections between Ethereum nodes, e.g., exploiting CVE-2020-20328 [21] and CVE-2016-5696 [10].

Note that the victim node, which we control, is running on the Ethereum testnet. By default, the configuration of the victim is the same as the node in Frankfurt (Table II). To simulate Eclipse [75] and Gethlighting [34] attacks, we install the vulnerable versions of Geth, v1.7.3 and v1.10.20, respectively. Additionally, we use Scapy to implement packet injection for DoS attacks, scanning for partition attacks, and sniffing for disturbing attacks. Finally, we use default implementations and the vulnerable Linux kernels for hijacking attacks [10], [21].

Furthermore, P2P networks are also attractive targets for traditional network attacks. Therefore, we replay 43 attack traffic in existing studies, to complement the attack against P2P networks, including: (i) Kitsune datasets [54], which contain attacks aimed at IoT devices [28]; (ii) Whisper datasets [24], which comprise both volumetric and stealthy attacks [17], [40], [47]; and (iii) HyperVision datasets [25], containing both traditional flooding attacks [18], [29], [46], [60], and various sophisticated attacks [21], [22], [51], [53], [58]. Finally, we replay the attack traffic datasets alongside eight benign traffic datasets collected at the eight nodes, using their original packet rates. Note that, we replay all attack traffic in a same dataset over the benign traffic dataset. For

TABLE II
DESCRIPTION OF REAL-WORLD DATASET COLLECTION

Location	Execution clients		Consensus Clients		Consensus Clients		Duration	FP Rates	Accuracy[1]	
	Software	Sync. Mode	Software	Sync. Mode	CPUs	Memory	(h)	(10^3 Alarms/h)	AURoC	F1-Score
DE, Frankfurt	Geth	Snap	Lighthouse	Checkpoint	16	64GB	3.25	13.01	0.9070	0.7896
JP, Tokyo	Nethermind	Snap	Lighthouse	Checkpoint	16	46GB	1.13	10.52	0.9959	0.9703
CN, HongKong	Besu	Fast	Lighthouse	Checkpoint	16	64GB	1.01	4.24	0.8708	0.5342
KR, Seoul	Erigon	Full	Lighthouse	Checkpoint	16	64GB	2.00	1.32	0.9333	0.7358
SG, Singapore	Reth	Full	Teku	Optimistic	32	128GB	1.66	0.84	0.9978	0.9837
US, Virginia	Geth	Snap	Nimbus	Checkpoint	32	128GB	1.66	1.30	0.9996	0.9976
CA, Toronto	Geth	Snap	Prysm	Optimistic	32	128GB	3.25	1.61	0.9748	0.8425
US, San Francisco	Geth	Snap	Teku	Checkpoint	32	128GB	1.66	0.70	0.9558	0.8522
Overall	-	-	-	-	-	-	1.95	4.19	0.9107	0.8193

[1] We train an K-Means model with CICFlowMeter features using the traffic collected on the collected traffic from these nodes.

TABLE III
DETECTION ACCURACY OF *tNeuron* FOR VARIOUS DETECTION METHODS ON VARIOUS DATASETS

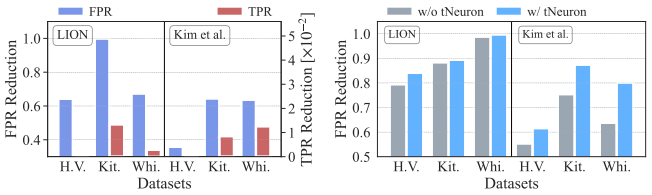
	Methods	FlowMeter-RF		FSC		Jaqen		N3IC		NetBeacon		FAE		Whisper		Overall	
	Datasets	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP
DoS	Conn. SYN	0.9994	-0.0125	0.4398	-0.0016	0.5597	-0.0112	0.7521	-0.0112	0.8820	-0.0112	0.8446	-0.1667	0.9498	-0.0472	0.7753	-0.0374
	Conn. RST	0.9649	-0.0014	0.5853	-0.0000	0.9151	-0.0112	0.8814	-0.0112	0.9781	-0.0125	0.9042	-0.1875	0.9486	-0.0395	0.8825	-0.0376
	Empty Conn.	0.9110	-0.0248	0.9805	0.0406	0.7831	-0.0353	0.9402	0.0286	0.9495	-0.0129	0.5612	-0.0247	0.9191	-0.0265	0.8635	-0.0079
	Gethlighting	0.6247	-0.0133	0.9999	-0.0140	0.7600	-0.0579	0.9240	0.0042	0.9999	0.0401	0.7084	0.0187	0.7200	-0.0146	0.8195	-0.0053
Disturb.	Delay-50%	0.7737	0.0231	0.9999	-0.0512	0.6707	-0.0494	0.8452	-0.0595	0.8635	-0.0592	0.8498	-0.0554	0.8580	-0.0196	0.8373	-0.0387
	Delay-80%	0.7644	-0.0264	0.7153	-0.0379	0.8129	0.0049	0.5353	-0.0296	0.9999	0.0228	0.7522	0.0449	0.7910	-0.0423	0.7673	-0.0091
	Pause-10s	0.7491	0.0574	0.9217	-0.0298	0.6809	-0.0244	0.6257	0.0114	0.9389	-0.0875	0.8680	-0.0721	0.7296	-0.0262	0.7877	-0.0245
	Pause-30s	0.9999	-0.0242	0.9067	-0.0930	0.9191	0.0024	0.7355	-0.0425	0.7006	-0.0090	0.8408	-0.0424	0.7111	0.0309	0.8305	-0.0254
Partition	Drop-10%	0.6971	-0.0007	0.6006	-0.0559	0.9999	0.0074	0.8057	0.0082	0.7732	-0.0108	0.9433	-0.0638	0.9999	-0.0676	0.8314	-0.0262
	Drop-20%	0.8485	-0.0551	0.9639	-0.0573	0.9999	-0.1008	0.8551	-0.0215	0.8511	0.0107	0.8866	0.0415	0.8785	-0.0220	0.8976	-0.0292
	Prefix	0.9999	0.0122	0.8534	-0.0947	0.6515	0.0421	0.9999	-0.0859	0.7176	0.0083	0.8924	-0.1057	0.7886	-0.0080	0.8433	-0.0331
	Eclipse	0.7398	-0.0125	0.4420	-0.0002	0.7887	-0.0071	0.7483	-0.0112	0.7168	-0.0072	0.8966	-0.0167	0.9549	-0.0208	0.7553	-0.0108
Hijack.	Hijack-DNS	0.9736	0.0270	0.9999	0.0188	0.9999	0.0545	0.9539	0.0556	0.8320	-0.0353	0.9704	-0.0842	0.8585	-0.0317	0.9412	0.0007
	Spoof-ACK	0.9784	-0.0111	0.5956	-0.0000	0.9045	0.0000	0.8644	0.0000	0.9122	-0.0000	0.8744	-0.0167	0.9444	-0.0542	0.8677	-0.0117
	Spoof-IPID	0.9310	-0.0000	0.7176	-0.0016	0.5105	0.0000	0.7585	0.0000	0.9122	0.0000	0.9043	-0.0303	0.9330	-0.0139	0.8096	-0.0065
	Average	0.8637	-0.0042	0.7815	-0.0252	0.7971	-0.0124	0.8150	-0.0110	0.8685	-0.0109	0.8465	-0.0507	0.8657	-0.0269	0.8340	-0.0202

[1] We highlight the best performance in ■ and the worst accuracy in ■.

the eight datasets, we use one for training and the remaining seven for testing, averaging the accuracy across these testing datasets. By default, experimental results are measured by using Ethereum attack traffic.

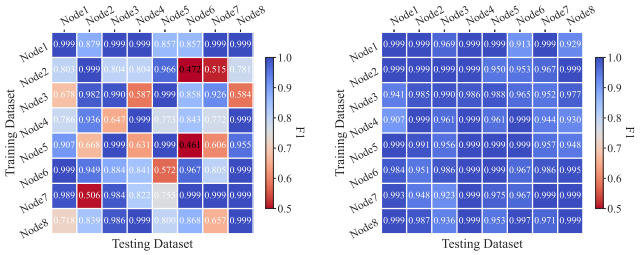
Baseline Methods. First, we collect alarms raised by seven state-of-the-art generic malicious traffic detection methods. Specifically, these methods extract various flow features [5], [24], [50], packet features [65], [80], frequency domain features [24], [26], and utilize various supervised and unsupervised machine learning (ML) algorithms with different settings. We deployed the open-source methods [24], [25], [54] directly and retrained their ML models. Meanwhile, we implemented closed-source methods that rely on specific network devices [65], [80]. Moreover, CICFlowMeter [11] is a widely-used tool that extracts flow-level features from traffic datasets. Since the tool only provides a set of features, we utilize random forests to classify the features, resulting in an established end-to-end traffic detection system, which we denote as FlowMeter-RF. In addition to the random forest model, we also other nine models served as benchmarks for robustness analysis in Section IV-E. Additionally, we used RFs to learn the traffic features extracted by Jaqen [50], which is originally a fixed rule-based method, to demonstrate the potential feasibility of *tNeuron* in identifying false positives raised by traditional rule-based detection.

Second, we validate that *tNeuron* can reduce false alarms for two state-of-the-art task-specific detection methods developed for Bitcoin. Specifically, we prototype the method developed by Kim et al. [42], [43], which analyzes statistical features of Bitcoin traffic. More precisely, we adapt the method for our Ethereum nodes by allowing the methods to analyze all the traffic from the nodes using its autoencoder model. Note that clock cycle based features are not available on the Ethereum nodes. Similarly, we prototype LION [20], which analyzes Bitcoin flow features using statistical methods. Particularly, we set $T_{\text{Test}} = 20$ minutes and $T_{\text{Train}} = 30$ minutes and use the other original parameters without modification, because the traces in our datasets are significantly shorter. Our experiments show that even if these task-specific methods can achieve high accuracy at one node, *tNeuron* can further improve their accuracy when deploying them to many nodes.



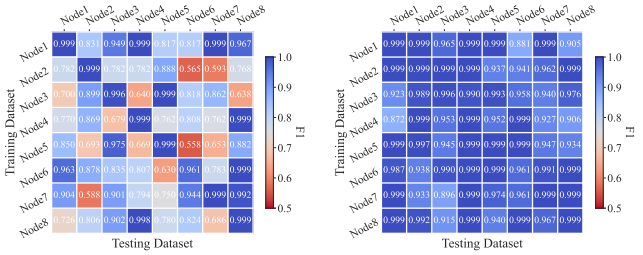
(a) FPR reductions. (b) Accuracy improvements.

Fig. 3. *tNeuron* can reduce FPs for task-specific detection systems.



(a) Disabled *tNeuron*. (b) Enabled *tNeuron*.

Fig. 4. Comparing the accuracy of FlowMeter-RF when detcting P2P DoS attacks with and without enabling *tNeuron*.



(a) Disabled *tNeuron*. (b) Enabled *tNeuron*.

Fig. 5. Comparing the accuracy of NetBeacon when detecting P2P connection hijacking attacks with and without enabling *tNeuron*.

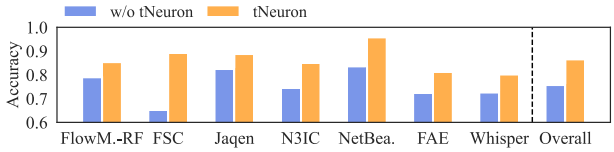


Fig. 6. *tNeuron* reduces FPs to improve detection accuracy for capturing various P2P network attacks.

Metrics: To assess the effectiveness of *tNeuron*, we primarily use the ratio of reduced false positives (R.FP). Also, we use reduced true positives (R.TP) to measure collateral damage on true positives, which should be minimal. Furthermore, we measure the increase in various accuracy metrics which are widely used to evaluate ML based traffic detection systems [15], [24], [54], [67], [81]. These metrics include area under the precision-recall curve (AUPRC) and receiver operating characteristic curve (AURoC), Matthews correlation coefficient (MCC), accuracy (Acc.), equal error rate (EER), precision (Pre.), F1- and F2-score.

B. Accuracy Evaluation

In general, *tNeuron* can reduce 83.40% FPs for the seven existing malicious traffic detection systems on the Ethereum P2P network traffic datasets, while only 2.02% TPs are misclassified by *tNeuron* as FP. By reducing massive FPs without significant damage on TPs, *tNeuron* can improve various accuracy metrics. Therefore, we conclude that *tNeuron* is able to deploy the trained traffic detection model from one node to other nodes in P2P networks.

Additionally, *tNeuron* reduces 87.53% FPs on existing public datasets, while the TPs misclassified by *tNeuron* as FP can be bounded by 1.51%. We mainly discuss the accuracy on P2P network traffic datasets in this section.

Reducing FPs Raised by Various Methods. First, the results presented in Table III demonstrate the effectiveness of *tNeuron* in detecting false positives (FPs). For widely used benchmarking detection methods such as FlowMeter-RF [11] and FSC [24], *tNeuron* can reduce FPs by 86.37% and 78.15%, respectively. Additionally, *tNeuron* can reduce FPs raised by supervised models, such as the tree model in NetBeacon [80], as well as FPs raised by unsupervised models, such as the K-Means model in FSC. Furthermore, *tNeuron* can identify FPs raised by methods deployed on various devices. For instance, *tNeuron* can reduce FPs by 97.04% for FAE deployed on x86 CPU and up to 99.99% for SmartNIC based methods. Moreover, *tNeuron* can detect FPs raised by methods that extract various features. Specifically, it can reduce FPs by 86.85% for flow feature based methods, 81.50% for packet-based features, and 86.57% for frequency features.

Reducing FPs Triggered by Various Traffic. Second, we demonstrate that *tNeuron* can effectively reduce false positives (FPs) on various datasets. In particular, we find that *tNeuron* can effectively reduce false alarms when detecting attacks targeting Ethereum. Specifically, when applying existing methods to detect DoS, Eclipse, and hijacking attacks, *tNeuron* reduces 88.25%, 75.53%, and 86.77% FPs, respectively, with a negligible 2.08% TPR decrease. This allows *tNeuron* to improve 14.62% AUC for these methods, enabling them to achieve 0.9034 AUC against the attacks targeting Ethereum.

For existing traffic datasets, *tNeuron* achieves 87.83% accuracy in identifying FPs (see Appendix A for details). Note that the accuracy on existing datasets, which focus on non-Ethereum attacks, is similar to the accuracy when handling Ethereum network attacks. Therefore, *tNeuron* does not significantly suffer from the dataset bias issue [3], i.e., overfitting to a particular dataset.

Reducing FPs for Task-Specific Detection. To complement the generic detection, we compare two state-of-the-art task-specific methods, in Figure 3. We observe that although these methods achieve perfect accuracy on a single node, *tNeuron* can further improve their accuracy by reducing FPs triggered by unseen traffic on nodes other than where the methods are initially trained. Specifically, *tNeuron* can reduce 64.02% - 99.44% of FPs raised by LION [20], and 35.90% - 64.48% of FPs raised by the method developed by Kim et al. [42], [43] (see Figure 3(a)). Consequently, *tNeuron* improves 0.90% - 5.47% AUC for LION and 10.07% - 20.42% AUC for Kim et al. , enabling these methods to achieve 0.9974 and 0.8745 AUC respectively, under the setting of multi-node deployment (see Figure 3(b)).

Side-Effects of Misclassification. *tNeuron* false positive (FP) reduction does not decrease the true positive rate (TPR), which could allow attackers to evade detection. As shown in Table V, *tNeuron* rarely misclassifies TPs as FPs, resulting in an 2.02% overall TPR decrease. In particular, the ratios of misclassified TPs approach zero for FlowMeter-RF [11], FSC [24], Jaqen [50], and N3IC [65]. By reducing FPs without decreasing TPs, *tNeuron* can significantly improve the performance of traffic detection systems. The benefits of using *tNeuron* will be further analyzed in the next section.

In summary, *tNeuron* can reduce FPs for various methods on various datasets because it treats these methods as black boxes. The construction of the Transformer model and meta-learning model does not interfere with the training of detection models. Moreover, *tNeuron* does not impose any assumptions on the distribution of traffic, making it applicable to various traffic datasets. In conclusion, *tNeuron* can independently reduce FPs raised by many methods when deploying these methods to analyze traffic that differs from the traffic in the datasets.

C. Improvement Evaluation

In this section, we demonstrate that by reducing FPs when deploying an ML model to different nodes, *tNeuron* can significantly improve the detection accuracy of various methods, i.e., 14.27% AURoC. We also use various metrics to confirm the improvements on nine different metrics (see Appendix B for details).

Improving Performance of Various Models. We plot the detailed detection accuracy of each node for Figure 4 and Figure 5. We observe that CICFlowMeter features can accurately capture attacks at the node where the training datasets are collected, i.e., achieving 0.9999 F1 (see Figure 4(a)). However, the model produces lower accuracy when detecting the same attack at the nodes other than where it is trained, i.e., the accuracy drops to 0.8267 F1. After we enable *tNeuron* to reduce false alarms, the accuracy on these nodes improves by 14.76% (0.9743 F1). We have similar observations when applying NetBeacon [80] to detect hijacking attacks (see Figure 5(a) and Figure 5(b)). Please note that we average the results by default to effectively present the data.

Additionally, Figure 9 indicate that the baseline methods, when deployed across multiple nodes, yield low detection accuracy ranging from 0.7832 to 0.9084. By enabling *tNeuron*, these baselines experience significant improvements in detection accuracy, achieving accuracy between 0.8633 and 0.9741. Specifically, Figure 7 compares the precision-recall curve (PRC) before and after applying *tNeuron* to enhance a model, where the area of the blue shadow indicates the improvements. It is observed that *tNeuron* can improve AUPRC by 3.64%, 12.15%, and 11.22% for three flow based methods, namely FlowMeter-RF [11], FSC [24], and Jaqen, when detecting stealthy and low-rate TCP attacks. Moreover, for packet based and frequency based methods, *tNeuron* can improve AUPRC by 0.1565 - 0.4554 and 0.0122 - 0.2932, respectively. Similarly, Figure 8 shows that *tNeuron* can improve AURoC by up to 17.59% and enable the detection model to identify many stealthy attacks that would have gone undetected without *tNeuron*. For instance, it can detect SQL server target attacks with NetBeacon [80] (see Figure 8(d)).

Real-World Case Study. We leverage *tNeuron* to facilitate the detection of real-world attacks targeting our five Ethereum nodes. Specifically, through Zeek network logs, we observe that five of our Ethereum nodes received TCP port scanning traffic originating from an address in Amsterdam, Netherlands. This scanning traffic consists of high-speed TCP SYN packets ranging from 57.21 to 354.61 PPS, which significant deviates from typical benign TCP application behaviors, where only a single SYN packet is generated to initialize a connection.

We use both attack and benign traffic from one node to train the FlowMeter-RF, and test the model on the other four nodes. From Figure 10, we observe that the model achieves 0.9871 AUC at the node where it is trained, yet it achieves only 0.8813 accuracy on the other nodes. When *tNeuron* is enabled to reduce false alarms, the model can achieve 0.9873 AUC at the nodes other than where it is trained. Note that we mainly replay existing datasets instead of using real-world attack traffic datasets, as we have only collected limited types of attacks.

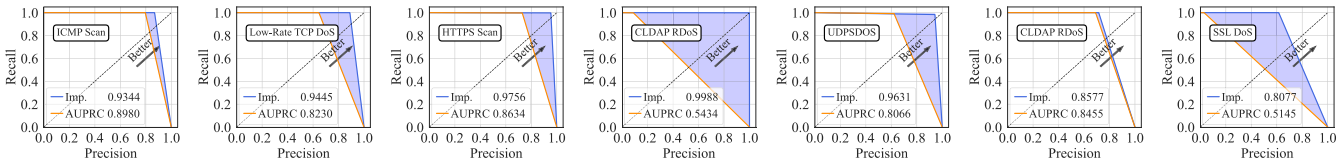
In conclusion, *tNeuron* can significantly improve the detection performance of various detection models by reducing most of the FPs when applying these models to detect attacks targeting nodes other than where they were trained. Moreover, the improvements are confirmed by various metrics, including PRC- and RoC-related metrics, accuracy, and MCC.

D. Transferability Evaluation

In this section, we validate the transferability of *tNeuron*, i.e., the effectiveness when enhancing different ML models under various settings.

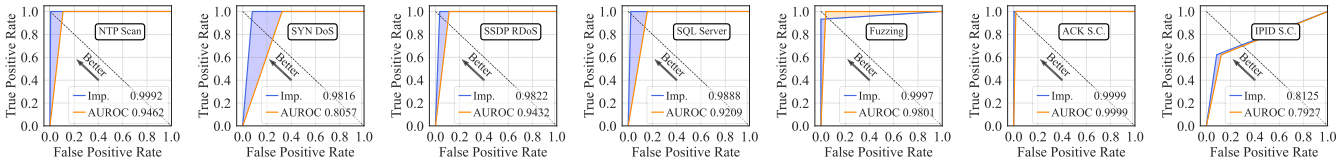
Reducing FPs Raised by Various Models. We first constructed a series of baselines by using different models to detect malicious traffic based on CICFlowMeter features [11]. Additionally, *tNeuron* can reduce 58.53% - 89.97% of FPs for eight widely used models [5], [8], [14], [24], [54], [63], [68], [80]. We observed that *tNeuron* can reduce 66.04% FPR for unsupervised models, which is significantly lower than for supervised models (e.g., reducing over 80% FPs for decision tree and linear classifier). This is because unsupervised models tend to generate more FPs [1], [27], [45]. Moreover, the maximum reduction rate of FPs ranges between 82.80% to 99.99%, while the average misclassification rate of TPs can be bounded by 3.25%. Therefore, *tNeuron* is a generic approach that can be used to adapt various ML models in P2P networks.

Reducing FPs under Various Settings. Secondly, we validated whether *tNeuron* is applicable to models with various hyper-parameters. For this purpose, we adjusted the number of trees and the depth of trees for the random forest model, ranging from 20 to 100 with a step length of 20. From Figure 11(a) and Figure 11(b), we found that for different values of depth, the improved AUPRC and AURoC ranged between 4.041% - 9.48% and 4.75% - 7.30%, respectively. Similarly, Figure 11(c) and Figure 11(d) showed that the variation of improved AURoC and AUPRC could be bounded by 1.30% and 1.14% for different numbers of trees. Moreover,



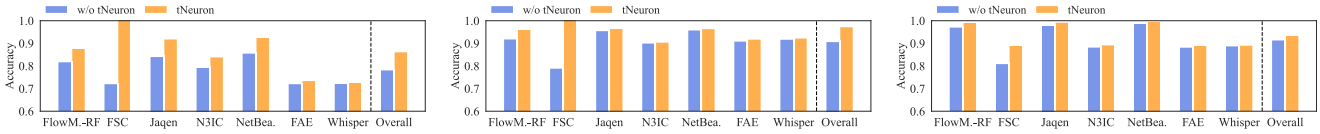
(a) FlowMeter-RF. (b) FSC. (c) Jaqen. (d) NetBeacon. (e) N3IC. (f) FAE. (g) Whisper.

Fig. 7. Comparing PRC of directly deploying ML model and applying *tNeuron* to enhance the performance of model by reducing FPs.



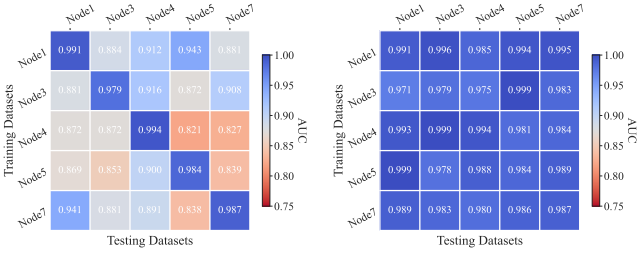
(a) FlowMeter-RF. (b) FSC. (c) Jaqen. (d) NetBeacon. (e) N3IC. (f) FAE. (g) Whisper.

Fig. 8. Comparing RoC of directly deploying ML model and applying *tNeuron* to enhance the performance of model by reducing FPs.



(a) HyperVision datasets. (b) Whisper datasets. (c) Kitsune datasets.

Fig. 9. Comparing accuracy of existing methods before and after enabling *tNeuron* for false alarm reduction.



(a) Disabled *tNeuron*. (b) Enabled *tNeuron*.

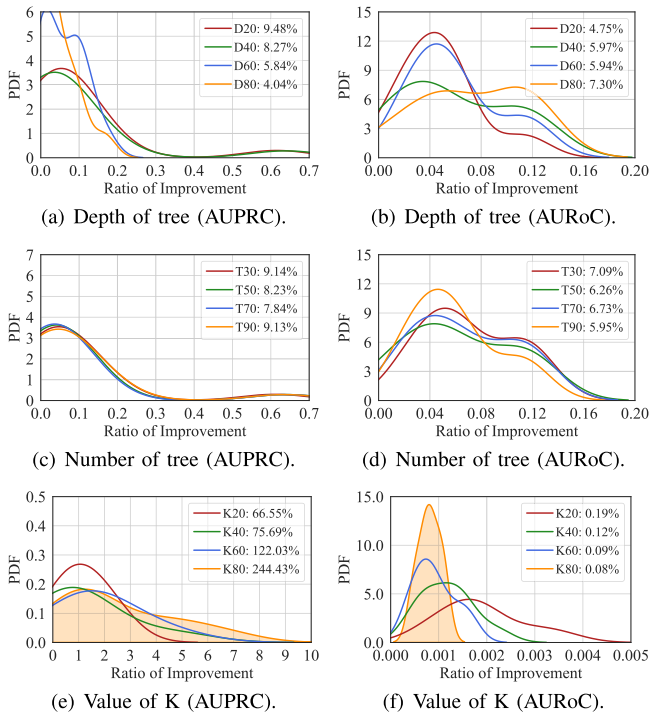
Fig. 10. Accuracy improvements of *tNeuron* using real-world attack traffic.

Figure 11(e) and Figure 11(f) indicated that *tNeuron* can also be applied to K-Means models with different values of K.

E. Robustness Evaluation

To demonstrate that attackers cannot easily exploit *tNeuron* to evade detection by mimicking benign users, we conducted experiments to validate its robustness against three evasion techniques based on recent studies [24], [26]: (i) traffic obfuscation, which involves injecting benign traffic into malicious flows; (ii) adaptive traffic rates, where attackers adjust packet rates to mimic benign flows; and (iii) manipulating flow features, where attackers manipulate packet lengths to generate benign traffic patterns. Note that, these adaptive attacks can evade many existing methods [5], [54], [65], [80] that do not raise alarms under these evasions. Therefore, we collected the alarms raised by three methods that are robust against the evasions and applied *tNeuron* to analyze these alarms.

Table IV shows that *tNeuron* reduces 94.40% - 96.48% FPs under the evasion attacks. Meanwhile, the incurred TPR decrease is bounded by 1.56%. By reducing the majority of FPs without significant TPR decrease, *tNeuron* significantly improves precision over the existing methods. Therefore, attackers cannot evade detection when *tNeuron* is applied.



(a) Depth of tree (AUPRC). (b) Depth of tree (AURoC).
(c) Number of tree (AUPRC). (d) Number of tree (AURoC).
(e) Value of K (AUPRC). (f) Value of K (AURoC).

Fig. 11. Distribution of accuracy improvement under various settings.

F. Ablation Study

We conduct ablation studies by individually disabling each key design feature. The results are shown in Figure 12. Initially, we replace the Transformer model with fully connected

TABLE IV
DETECTION ACCURACY OF *tNeuron* UNDER EVASION ATTACKS

Evasions	Faked Length		Slowdown		Obfuscation		Overall	
Datasets	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP
Chargen	0.9574	0.0000	0.9524	-0.0833	0.9894	0.0000	0.9664	-0.0278
CLDAP	0.9737	0.0000	0.9794	0.0000	0.9636	0.0000	0.9722	0.0000
DNS	1.0000	0.0000	1.0000	0.0000	0.9765	0.0000	0.9922	0.0000
MemC.	0.9762	0.0000	0.9485	0.0000	0.9524	0.0000	0.9590	0.0000
NTP	0.9889	0.0000	0.9726	0.0000	0.9512	0.0000	0.9709	0.0000
SSDP	0.8615	0.0000	0.9730	0.0000	0.9785	0.0000	0.9377	0.0000
Avg	0.9596	0.0000	0.9710	-0.0139	0.9686	0.0000	0.9664	-0.0046
SSH	0.9787	0.0000	0.9545	0.0000	0.9016	0.0000	0.9449	0.0000
DNS	1.0000	0.0000	1.0000	0.0000	0.8947	0.0000	0.9649	0.0000
HTTP	1.0000	0.0000	0.9242	0.0000	1.0000	0.0000	0.9747	0.0000
NTP	1.0000	0.0000	0.9545	0.0000	0.9512	0.0000	0.9686	0.0000
SqlServ.	1.0000	0.0000	0.9844	0.0000	1.0000	0.0000	0.9948	0.0000
HTTP	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
Avg	0.9964	0.0000	0.9696	0.0000	0.9579	0.0000	0.9747	0.0000
SYN	0.9865	0.0000	1.0000	0.0000	1.0000	0.0000	0.9955	0.0000
RST	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
UDP	1.0000	-0.2500	0.9865	0.0000	0.9800	0.0000	0.9888	-0.0833
Avg	0.9955	-0.0833	0.9955	0.0000	0.9933	0.0000	0.9948	-0.0278

[1] We highlight the best accuracy in ■ and the worst accuracy in ■.
[2] We mark - for the methods when the AUC is lower than 0.50.

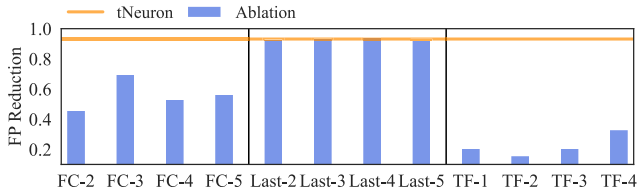


Fig. 12. Ablation studies for *tNeuron*.

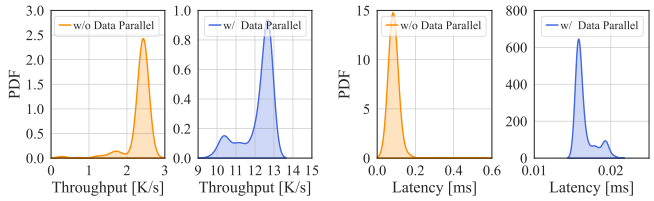
layers (comprising L_{Expand} hidden states) that learn the CICFlowMeter [11] features. We find that using 2 - 5 layers of fully connected layers can only reduce 46.24% - 69.96% FPs on HyperVision datasets [25], which is significantly lower than the 93.14% reduction achieved by the Transformer model. The reason is that fully connected layers cannot effectively extract fine-grained sequential features to provide rich patterns for neuron activation analysis.

Second, we analyze the activation patterns of the last 2 - 5 layers instead of only the last layer. The results demonstrate that analyzing additional layers does not significantly improve accuracy. Specifically, analyzing 2 - 4 layers can reduce 93.44% - 94.56% FPs, which is similar to using just the last layer. Moreover, analyzing the last five layers achieves only a 92.65% reduction, because activation patterns in the earlier layers are too complex for analysis, and thus we use the last layer for neuron activation analysis.

Finally, we reduce the number of layers (N_L) for the Transformer model. The results indicate that using fewer layers cannot effectively generate neuron activation patterns, and thus cannot significantly reduce FPs.

G. Performance Evaluation

We execute *tNeuron* by using four NVIDIA Tesla V100 GPUs, and measure the performance of the *tNeuron* with



(a) Detection throughput. (b) Detection latency.

Fig. 13. Detection latency and throughput of *tNeuron*.

and without data parallel. Figure 13(a) shows that *tNeuron* can achieve 0.283 - 2.674 and 9.733K - 12.820K alarm/s processing throughput when data parallel is disabled and enabled, respectively. Meanwhile, average detection latency is 0.0164s when data parallel is enabled (see Figure 13(b)). Therefore, we conclude that *tNeuron* can achieve real-time detection with a high processing capacity, making it capable of timely handling massive FPs raised by many P2P nodes. This is because, we utilize light-weight statistical ML to model neuron activation patterns, which does not incur computation intensive operations. Meanwhile, we use hyper-parameter L_{Max} to control the number of analyzed packets which limits the overhead of the Transformer models.

V. RELATED WORK

ML Based Attack Traffic Detection. ML based detection has been shown to achieve higher accuracy than signature-based detection [50], [78]. Moreover, ML algorithms have the ability to unknown attacks [24], [54]. For generic detection approaches, Fu et al. utilized the frequency domain features [24]. Zhou et al. [80] developed NetBeacon to implement ML models on programmable switches. Panda et al. [59] built flow based detection upon SmartNICs. Moreover, HyperVision utilized graphs to detect encrypted malicious traffic [25]. For task specific detection, Nelms et al. [55], Invernizzi et al. [36], and Bilge et al. [8] detected traffic generated by malware campaigns. Bartos et al. [6], Tang et al. [67], and Dodia et al. [14] detected Web attack traffic. Wichtlhuber et al. [74] and Wagner et al. [72] detected DDoS traffic at IXP-level. Dodia et al. [69] and Sharma et al. [63] captured IoT attack traffic.

In this work, we mainly focus on generic detection methods [5], [24], [26], [35], [80], which are more profitable to deploy. Recently, task-specific attack traffic detection systems have also been developed for cryptocurrency systems such as Bitcoin [20], [42], [43]. For instance, Fan et al. [20] developed LION, a lightweight detection method that analyzes features of Bitcoin messages using statistical methods to capture various advanced attacks targeting a Bitcoin node. Moreover, Kim et al. [42], [43] developed an autoencoder based method that analyzes various statistical features extracted from Bitcoin flows. Unlike these methods, *tNeuron* aims to enhance the performance for these methods when deploying them across numerous P2P nodes. Note that, traffic classification, which classifies benign traffic [4], [61], is entirely different to malicious traffic detection systems focused in this paper. These

TABLE V
DETECTION ACCURACY OF *tNeuron* FOR VARIOUS DETECTION METHODS ON EXISTING DATASET

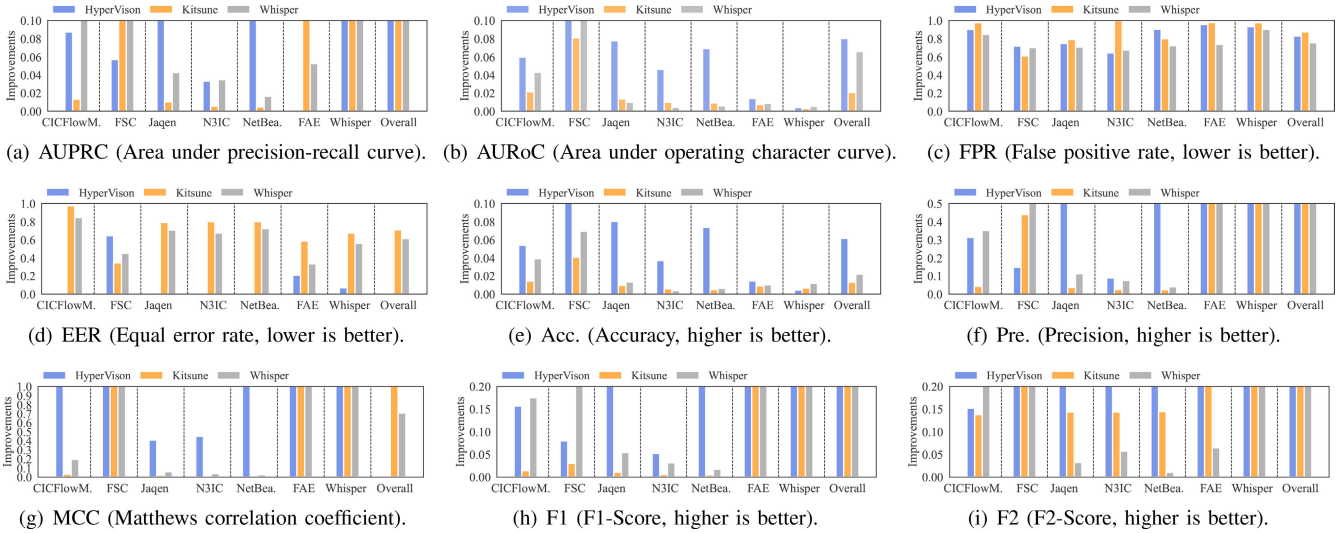
	Methods	FlowM.-RF		FSC		Jaqen		N3IC		NetBeacon		FAE		Whisper		Overall	
	Datasets	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP	R.FP	R.TP
HyperVision Datasets	DNS [1]	0.9956	0.0000	0.7493	0.0000	0.9860	0.0000	0.5087	0.0000	0.9998	0.0000	0.9571	-0.1250	0.8944	0.0000	0.8701	-0.0179
	HTTP	0.8360	0.0000	0.5171	-0.0000	0.8690	0.0000	0.4853	-0.0003	0.8716	0.0000	0.9644	-0.1111	0.9034	0.0000	0.7781	-0.0159
	SqlServ.	0.8262	0.0000	0.5172	0.0000	0.8615	0.0000	0.4872	-0.0002	0.8631	0.0000	0.9635	0.0000	0.9086	0.0000	0.7753	-0.0000
	ICMP	0.4103	0.0000	0.4524	0.0000	0.4922	0.0000	0.7466	-0.0001	0.4758	0.0000	0.9645	0.0000	0.8832	0.0000	0.6321	-0.0000
	SSH	0.4627	0.0000	0.4446	0.0000	0.3187	0.0000	0.0274	0.0000	0.2978	0.0000	0.9615	-0.1000	0.9233	-0.1000	0.4909	-0.0286
	HTTPS	0.6994	0.0000	0.5001	0.0000	0.7595	0.0000	0.4767	-0.0001	0.7536	0.0000	0.9649	-0.2000	0.9206	0.0000	0.7250	-0.0286
	NTP	0.9957	0.0000	0.7582	0.0000	0.9851	0.0000	0.4864	0.0000	0.9997	0.0000	0.9522	-0.1111	0.9058	0.0000	0.8690	-0.0159
	Chargen	0.9986	0.0000	0.8601	0.0000	0.6884	0.0000	0.7995	-0.0000	0.9999	0.0000	0.9404	0.0000	0.9435	0.0000	0.8901	0.0000
	Memcache.	1.0000	0.0000	0.9331	0.0000	0.3744	0.0000	0.7926	0.0000	0.9999	0.0000	0.9689	0.0000	0.9603	0.0000	0.8613	0.0000
	CLDAP	0.9982	0.0000	0.8613	0.0000	0.6883	0.0000	0.7895	0.0000	0.9999	0.0000	0.9378	0.0000	0.9338	0.0000	0.8870	0.0000
	RIP	0.9987	0.0000	0.9610	0.0000	0.6878	0.0000	0.8031	0.0000	0.9999	0.0000	0.9341	0.0000	0.9368	0.0000	0.9031	0.0000
	SSDP	0.9986	0.0000	0.0134	0.0000	0.6878	0.0000	0.7946	-0.0025	0.9999	0.0000	0.9388	-0.1818	0.9170	0.0000	0.7643	-0.0263
	DNS	0.9986	0.0000	0.8474	-0.0010	0.6884	0.0000	0.8010	0.0000	0.9999	0.0000	0.9377	0.0000	0.9330	0.0000	0.8866	-0.0001
	NTP	0.9985	0.0000	0.8586	0.0000	0.6878	0.0000	0.8023	0.0000	0.9999	0.0000	0.9341	0.0000	0.9366	0.0000	0.8883	0.0000
	UDP	1.0000	0.0000	0.9918	0.0000	0.6981	0.0000	0.8149	-0.0001	1.0000	0.0000	1.0000	-	0.9636	-0.2000	0.9241	-0.0286
	ICMP	1.0000	0.0000	0.9973	0.0000	1.0000	0.0000	0.4201	-0.0007	1.0000	0.0000	1.0000	- [2]	1.0000	-	0.9168	-0.0001
	SYN	0.9989	0.0000	0.7570	0.0000	0.9901	0.0000	0.7606	0.0000	1.0000	0.0000	0.9495	0.0000	0.9470	0.0000	0.9147	0.0000
	RST	0.9988	0.0000	0.9003	0.0000	0.9893	0.0000	0.7723	0.0000	0.9999	0.0000	0.9510	0.0000	0.9548	0.0000	0.9381	0.0000
	Average	0.9008	0.0000	0.7178	-0.0001	0.7474	0.0000	0.6427	-0.0002	0.9034	0.0000	0.9567	-0.0461	0.9314	-0.0167	0.8286	-0.0090
Kitsune Datasets	SSL DoS	1.0000	0.0000	0.4885	0.0000	0.9684	0.0000	1.0000	0.0000	0.9983	0.0000	0.9466	0.0000	0.9813	0.0000	0.9119	0.0000
	SSDP DoS	0.9990	-0.0000	0.7477	-0.0002	1.0000	-0.0000	1.0000	-0.0000	1.0000	-0.0000	1.0000	0.0000	0.9936	0.0000	0.9629	-0.0000
	OS FP.	1.0000	0.0000	0.4438	-0.0000	0.0000	-0.0000	1.0000	0.0000	0.0000	0.0000	0.9502	0.0000	0.9847	0.0000	0.6255	0.0000
	TCP SYN	0.9900	0.0000	0.3734	0.0000	0.9867	0.0000	1.0000	0.0000	1.0000	0.0000	0.9949	-0.1250	0.9928	0.0000	0.9054	-0.0179
	Fuzzing	0.8850	-0.0669	0.9953	-0.0114	0.9985	-0.0669	0.9970	-0.0669	0.9976	-0.0669	1.0000	0.0000	0.9226	-0.1250	0.9709	-0.0577
	Average	0.9748	-0.0134	0.6097	-0.0023	0.7907	-0.0134	0.9994	-0.0134	0.7992	-0.0134	0.9783	-0.0250	0.9750	-0.0250	0.8753	-0.0151
Whisper Datasets	OS Scan	1.0000	0.0000	0.4714	-0.0000	0.0000	-0.0000	0.0000	0.0000	0.0488	0.0000	0.5989	0.0000	0.9021	0.0000	0.4316	0.0000
	SYN DoS	1.0000	0.0000	0.3212	0.0000	0.9576	0.0000	1.0000	0.0000	0.7857	0.0000	0.6947	0.0000	0.9239	0.0000	0.8119	0.0000
	UDP DoS	0.8825	-0.0083	1.0000	0.0000	0.3873	-0.0083	0.9009	-0.0083	0.9742	-0.0083	0.7992	0.0000	0.9389	-0.0370	0.8404	-0.0100
	TLS Scan	0.7930	-0.0007	0.4371	-0.0011	1.0000	-0.0025	0.2646	0.0000	0.2641	0.0000	0.8242	-0.1000	0.9113	-0.0500	0.6420	-0.0220
	SSH DoS	1.0000	0.0000	0.4815	0.0000	0.8057	0.0000	0.9785	0.0000	0.9990	0.0000	0.5828	0.0000	0.9077	0.0000	0.8222	0.0000
	Fuzzing	0.8778	-0.0296	0.9953	0.0000	1.0000	-0.0428	0.9985	-0.0296	0.9993	-0.0428	0.6185	0.0000	0.8999	0.0000	0.9128	-0.0207
	SSDP DoS	0.9998	-0.0000	0.7289	-0.0001	1.0000	-0.0000	0.9998	-0.0000	0.9998	-0.0000	0.6258	0.0000	0.9029	0.0000	0.8939	-0.0000
	IPID SC	0.9977	0.0000	0.9830	0.0000	0.0395	0.0000	1.0000	0.0000	0.3158	0.0000	0.7758	0.0000	0.9066	-0.2000	0.7169	-0.0286
	ACK SC	0.4455	0.0000	0.3999	0.0000	1.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.8309	0.0000	0.8993	0.0000	0.6537	0.0000
	IPID Scan	0.0024	0.0000	0.9821	0.0000	0.0000	0.0000	0.0040	0.0000	0.0084	0.0000	0.7700	0.0000	0.8691	0.0000	0.3766	0.0000
	LRDoS 0.2	0.9999	0.0000	0.7969	0.0000	0.9999	0.0000	0.9167	0.0000	1.0000	0.0000	0.8752	0.0000	0.9030	-0.0833	0.9274	-0.0119
	LRDoS 0.5	1.0000	0.0000	0.7721	0.0000	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000	0.8802	0.0000	0.9090	0.0000	0.9373	0.0000
	LRDoS 1.0	1.0000	0.0000	0.7540	0.0000	1.0000	0.0000	0.7000	0.0000	1.0000	0.0000	0.7134	0.0000	0.8625	-0.2222	0.8614	-0.0317
	Average	0.8460	-0.0030	0.7018	-0.0001	0.7069	-0.0041	0.6741	-0.0029	0.7227	-0.0039	0.7377	-0.0077	0.9028	-0.0456	0.7560	-0.0096

[1] We use the data tags in the origin works [25], [26], [55] to denote the different attacks, and highlight the best performance in ■ and the worst accuracy in ■.
[2] FAE and Whisper can not detect these attacks, their average AUC $\leq$ 0.5 (i.e., worse than random guess).

methods aim to leak privacy by employing ML models to infer the access patterns of websites [61], usages of mobile apps [4], DNS queries [64], device fingerprints [5], and geological locations [35].

Enhancing ML Based Security Applications. To the best of our knowledge, none of the existing work aims to automatically FPs [3], [66], even though many studies require manually identified FPs to improve the performance of security applications [5], [25], [35], [80]. In general, enhancing the performance of these applications involves using access lists or model retraining, for access list based method, Tegeler et al. [68] crafted fixed rules based on known FPs to reduce similar FPs. Bilge et al. [8] used a similar approach to construct rules. For retraining based methods, Du et al. [15] incrementally retrained the DNN for log-level anomaly detection using FPs reported by user feedback and mitigated the catastrophic forgetting issues [16]. In summary, these methods only reduce FPs based on manually identified FPs, which are different from the automatic FP identification proposed in this paper. Similarly, many other ML based security applications suffer from high FPR issues. They also utilize model retraining [12], [31], [49] or fixed rules [38] based on manually identified FPs to further enhance the security applications.

Common Issues in ML Based Security Applications. Arp et al. [3] studied the issues in ML based security systems and underscored the high FPR issue [15], [54]. They disclosed that many security applications other than ML based traffic detection systems also suffer from high false positive issues [38], [49], [77]. Recently, Alahmadi et al. [1] showed that deploying ML based systems in SOCs is impeded by the high FPR issue, which is almost impossible to leverage human labor to inspect each alarm. Moreover, Sommer and Paxson [66] analyzed the reason for the low usability of ML based security systems by real-world survey, where many



(a) AUPRC (Area under precision-recall curve). (b) AURoC (Area under operating character curve). (c) FPR (False positive rate, lower is better).

(d) EER (Equal error rate, lower is better). (e) Acc. (Accuracy, higher is better). (f) Pre. (Precision, higher is better).

(g) MCC (Matthews correlation coefficient). (h) F1 (F1-Score, higher is better). (i) F2 (F2-Score, higher is better).

Fig. 14. *tNeuron* is capable of improving detection accuracy for many different methods.

human experts complain about the high false positive issues. Additionally, Han et al. [30], Jacobs et al. [37], and Wei et al. [73] addressed the explainability issues. These works aim to develop tools to facilitate model retraining, instead of reducing FP alarms in realtime, and are thus orthogonal to this work.

VI. Conclusion

To effectively deploy ML based attack traffic detection systems to protect distributed networks, we present *tNeuron* that automatically identifies FPs triggered by unseen traffic via neuron activation pattern analysis, such that it significantly improve the performance on various nodes. Specifically, we construct a shadow model with Transformer encoders to extract the knowledge of existing attack traffic detection systems. Afterward, we train a model that learns how to classify FPs among alarms raised by ML models according to neuron activation patterns of the shadow model. Our experiments on real Ethereum nodes show that *tNeuron* can reduce 64% $\sim$ 95% FP for seven state-of-the-art ML based attack detection system, when detecting 51 kinds of network attacks, thereby significantly improving detection accuracy in nine different metrics. In addition, *tNeuron* is robust against various adversarial examples constructed by existing evasion attacks. Besides, it achieves real-time detection and is capable of handling massive FPs generated by many nodes in large-scale distribution networks.

Appendic A
Detailed Accuracy of Traditional Attacks

Reducing FPs Raised by Various Methods. First, the results presented in Table V demonstrate the effectiveness of *tNeuron* in detecting false positives (FPs). For widely used benchmarking detection methods such as FlowMeter-RF [11] and FSC [24], *tNeuron* can reduce FPs by 84.60% - 90.08% and 60.97% - 70.18%, respectively. Additionally, *tNeuron* can reduce FPs raised by supervised models, such as the tree model in NetBeacon [80], as well as FPs raised by unsupervised models, such as the K-Means model in FSC. Furthermore, *tNeuron* can identify FPs raised by methods deployed on various devices. For instance, *tNeuron* can reduce FPs by 95.67% for FAE deployed on x86 CPU and up to 99.94% for SmartNIC based methods. Moreover, *tNeuron* can detect FPs raised by methods that extract various features. Specifically, it can reduce FPs by 71.87% - 90.08% for flow feature based methods, 90.34% - 99.94% for packet-based features, and 97.53% - 90.28% for emerging frequency features.

Reducing FPs Triggered by Various Traffic. Second, we demonstrate that *tNeuron* can effectively reduce false positives (FPs) on various datasets. On the HyperVision datasets, *tNeuron*-enhanced methods can capture attacks with high or low packet rates. For example, *tNeuron* can reduce FPs by 87.01% on DNS Amplification attacks that flood volumes of traffic to users [40], while detecting stealthy SSH scanning attacks that aim to crack user passwords [39]. On the Whisper datasets [26], *tNeuron* facilitates existing methods in detecting various advanced attacks. For instance, it can reduce FPs by 71.69%, 65.37%, and 93.73% for CVE-2020-20328 [21] IPID-based TCP side-channel attacks, ACK rate-limit-based side-channel attacks [10], and link flooding attacks (LFA) [41], [51], respectively. By analyzing the performance using different datasets, we eliminate the impact of dataset bias, which is critical for evaluating ML based systems [3].

Appendix B
Detailed Improvements on Other Metrics

Improving Performance in Various Metrics. We further confirm the improvements by using nine metrics. Figure 14 illustrates the improvements by averaging the accuracy of multiple methods on the datasets. From the figure, we observe that *tNeuron* improves various PRC-related metrics, including AUPRC, precision, F1, and F2, by 11.12%, 55.23%, 23.10%, and 26.11%, respectively, compared to the original performance. Additionally, *tNeuron* improves RoC-related metrics as well as PRC-related metrics. Specifically, it can improve AUROC by 2.01% - 8.12% on different datasets while

decreasing EER by up to 72.59%. Moreover, *tNeuron* improves MCC by over one time.

REFERENCES

[1] B. A. Alahmadi et al., "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 2783–2800.

[2] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking Bitcoin: Routing attacks on cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 375–392.

[3] D. Arp et al., "Dos and don'ts of machine learning in computer security," in *Proc. 31st USENIX Secur. Symp. (USENIX Secur.)*, 2022, pp. 3971–3988.

[4] A. Bahramali, A. Houmansadr, R. Soltani, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[5] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[6] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 807–822.

[7] H. L. J. Bijmans, T. M. Booij, and C. Doerr, "Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking," in *Proc. CCS*, 2019, pp. 449–464.

[8] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, Dec. 2012, pp. 129–138.

[9] Bitcoin. *A Peer-to-Peer Electronic Cash System*. Accessed: Jul. 2022. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[10] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path TCP exploits: Global rate limit considered dangerous," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 209–225.

[11] CIC. *CICFlowMeter: A Network Traffic Flow Generator and Analyser*. Accessed: Jul. 2022. [Online]. Available: https://www.unb.ca/cic/research/applications.html

[12] X. Deng et al., "Robust multi-tab website fingerprinting attacks in the wild," in *Proc. IEEE Symp. Secur. Privacy*, May 2023, pp. 1005–1022.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186.

[14] P. Dodia, M. AlSabah, O. Alrawi, and T. Wang, "Exposing the rat in the tunnel: Using traffic analysis for tor-based malware detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 875–889.

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

[16] M. Du, Z. Chen, C. Liu, R. Oak, and D. Song, "Lifelong anomaly detection through unlearning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1283–1297.

[17] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proc. 22nd USENIX Secur. Symp. (USENIX Secur.)*, 2013, pp. 605–620.

[18] Z. Durumeric, M. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *Proc. 23rd USENIX Conf. Secur. Symp.*, 2014, pp. 65–78.

[19] Ethereum. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Accessed: Jul. 2022. [Online]. Available: https://ethereum.org/en/

[20] W. Fan et al., "Lightweight and identifier-oblivious engine for cryptocurrency networking anomaly detection," *IEEE Trans. Depend. Secure Comput.*, vol. 20, no. 2, pp. 1302–1318, Mar. 2023.

[21] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path TCP exploits of the mixed IPID assignment," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds., 2020, pp. 1323–1335.

[22] X. Feng et al., "Off-path network traffic manipulation via revitalized ICMP redirect attacks," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 2619–2636.

[23] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.

[24] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3431–3446.

[25] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.

[26] C. Fu, Q. Li, M. Shen, and K. Xu, "Frequency domain feature based robust malicious traffic detection," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 452–467, Feb. 2023.

[27] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ML-based malicious traffic detection: Reducing majorities of false positive alarms," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 1005–1019.

[28] H. Griffioen and C. Doerr, "Examining Mirai's battle over the Internet of Things," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2020, pp. 743–756.

[29] H. Griffioen, K. Oosthoek, P. van der Knaap, and C. Doerr, "Scan, test, execute: Adversarial tactics in amplification DDoS attacks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 940–954.

[30] D. Han et al., "DeepAID: Interpreting and improving deep learning-based anomaly detection in security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3197–3217.

[31] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[32] M. E. Harrington, "Neurobiological studies of fatigue," *Prog. Neurobiol.*, vol. 99, no. 2, pp. 93–105, Nov. 2012.

[33] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 129–144.

[34] H. Heo, S. Woo, T. Yoon, M. S. Kang, and S. Shin, "Partitioning Ethereum without eclipsing it," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.

[35] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3366–3383.

[36] L. Invernizzi et al., "Nazca: Detecting malware distribution in large-scale networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–16.

[37] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Erreira, A. Gupta, and L. Z. Granville, "AI/ML for network security: The emperor has no clothes," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2022, pp. 1537–1551.

[38] S. T. Jan et al., "Throwing darts in the dark? Detecting bots with limited data using neural data augmentation," in *Proc. 41st IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1190–1206.

[39] M. Javed and V. Paxson, "Detecting stealthy, distributed SSH bruteforcing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 85–96.

[40] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 100–113.

[41] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, May 2013, pp. 127–141.

[42] J. Kim et al., "Anomaly detection based on traffic monitoring for secure blockchain networking," in *Proc. IEEE Int. Conf. Blockchain Cryptocurr. (ICBC)*, May 2021, pp. 1–9.

[43] J. Kim et al., "A machine learning approach to anomaly detection based on traffic monitoring for secure blockchain networking," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 3, pp. 3619–3632, Sep. 2022.

[44] A. M. Kirillov et al., "Segment anything," in *Proc. ICCV*, Oct. 2023, pp. 3992–4003.

[45] F. B. Kokulu et al., "Matched and mismatched SOCs: A qualitative study on security operations center issues," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, London, U.K., Nov. 2019, pp. 1955–1970.

[46] D. Kopp, M. Wichtlhuber, I. Poese, J. Santanna, O. Hohlfeld, and C. Dietzel, "DDoS hide & seek: On the effectiveness of a booter services takedown," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 65–72.

[47] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, Aug. 2003, pp. 75–86.

[48] H. Lee, H. Lee, J. Shin, E. Yang, T. M. Hospedales, and S. J. Hwang, "Online hyperparameter meta-learning with hypergradient distillation," in *Proc. ICLR*, Jan. 2021, pp. 1–17.

[49] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1777–1794.

[50] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. USENIX Secur. Symp.*, 2021, pp. 3829–3846.

[51] X. Luo and R. K. C. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proc. NDSS*, Jan. 2005, pp. 1–19.

[52] K. Man, X. Zhou, and Z. Qian, "DNS cache poisoning attack: Resurrections with side channels," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3400–3414.

[53] R. Merget et al., "Scalable scanning and automatic classification of TLS padding Oracle vulnerabilities," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1029–1046.

[54] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[55] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "WebWitness: Investigating, categorizing, and mitigating malware download paths," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 1025–1040.

[56] Y. Nir et al., "Selective neuronal lapses precede human cognitive lapses following sleep deprivation," *Nature Med.*, vol. 23, no. 12, pp. 1474–1480, Dec. 2017.

[57] A. Noroozian et al., "Platforms in everything: Analyzing ground-truth data on the anatomy and economics of bullet-proof hosting," in *Proc. 28th USENIX Secur. Symp. (USENIX Secur.)*, Jan. 2019, pp. 1341–1356.

[58] A. Oest et al., "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 2039–2056.

[59] S. Panda, Y. Feng, S. G. Kulkarni, K. K. Ramakrishnan, N. Duffield, and L. N. Bhuyan, "SmartWatch: Accurate traffic analysis and flow-state tracking for intrusion prevention using SmartNICs," in *Proc. 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2021, pp. 60–75.

[60] P. Richter and A. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 144–157.

[61] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[62] M. Saad, S. Chen, and D. Mohaisen, "SyncAttack: Double-spending in Bitcoin without mining power," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 1668–1685.

[63] R. A. Sharma, I. Sabane, M. Apostolaki, A. Rowe, and V. Sekar, "Lumen: A framework for developing and evaluating ML-based IoT network anomaly detection," in *Proc. 18th Int. Conf. Emerg. Netw. Exp. Technol.*, New York, NY, USA, Nov. 2022, pp. 59–71.

[64] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted DNS –> privacy? A traffic analysis perspective," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[65] G. Siracusano et al., "Re-architecting traffic analysis with neural network interface cards," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 513–533.

[66] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Security Privacy*, May 2010, pp. 305–316.

[67] R. Tang et al., "ZeroWall: Detecting zero-day web attacks through encoder–decoder recurrent neural networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 2479–2488.

[68] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, Dec. 2012, pp. 349–360.

[69] E. Tekiner, A. Acar, and S. Uluagac, "A lightweight IoT cryptojacking detection mechanism in heterogeneous smart home networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–15.

[70] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against Bitcoin peer-to-peer network," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 894–909.

[71] M. Tran, A. Shenoi, and M. S. Kang, "On the routing-aware peering against network-eclipse attacks in Bitcoin," in *Proc. Secur. USENIX*, Jan. 2021, pp. 1253–1270.

[72] D. Wagner et al., "United we stand: Collaborative detection and mitigation of amplification DDoS attacks at scale," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 970–987.

[73] F. Wei et al., "Xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 4337–4354.

[74] M. Wichtlhuber et al., "IXP scrubber: Learning from blackholing traffic for ML-driven DDoS detection at scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2022, pp. 707–722.

[75] K. Wüst and A. Gervais. *Ethereum Eclipse Attacks*. Accessed: Jul. 2022. [Online]. Available: https://research-collection.ethz.ch/bitstream/handle/20.500.11850/121310/eth-49728-01.pdf

[76] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *Proc. 29th USENIX Secur. Symp. (USENIX Secur.)*, Jan. 2020, pp. 2039–2056.

[77] L. Yang et al., "CADE: Detecting and explaining concept drift samples for security applications," in *Proc. 30th USENIX Secur. Symp.*, Vancouver, BC, Canada, 2021, pp. 2327–2344.

[78] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[79] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu, "Realtime DDOS defense using COTS SDN switches via adaptive correlation analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, pp. 1838–1853, 2018.

[80] G. Zhou et al., "NetBeacon: An efficient design of intelligent network data plane," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 6203–6220.

[81] S. Zhu et al., "You do (not) belong here: Detecting DPI evasion attacks with context learning," in *Proc. 16th Int. Conf. Emerg. Netw. EXperiments Technol.*, Nov. 2020, pp. 183–197.

Songsong Xu received the Ph.D. degree from Tsinghua University. His research interests include P2P applications, blockchain, and security.

Chuanpu Fu (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with Tsinghua University. His research interests include machine learning for security and network security.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, particularly in internet security. He is an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURITY COMPUTING, ACM TOPS, and ACM DTRAP.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 200 technical articles and holds more than ten U.S. patents in the research areas of next-generation internet, blockchain systems, the Internet of Things, and network security. He is a member of ACM. He has won the IWQoS 2024 Best Paper Award and the Distinguished Paper Award from USENIX Security 2023 and 2024.