Fall 2023

Detecting the Onion Routing Traffic in Real-Time by using Reinforcement Learning

Dazhou Liu
San Jose State University

Recommended Citation
Liu, Dazhou, "Detecting the Onion Routing Traffic in Real-Time by using Reinforcement Learning" (2023). *Master's Theses*. 5459.
DOI: https://doi.org/10.31979/etd.jham-bb2s
https://scholarworks.sjsu.edu/etd_theses/5459

DETECTING THE ONION ROUTING TRAFFIC IN REAL-TIME BY USING REINFORCEMENT LEARNING

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Dazhou Liu

December 2023

The Designated Thesis Committee Approves the Thesis Titled

DETECTING THE ONION ROUTING TRAFFIC IN REAL-TIME BY USING REINFORCEMENT LEARNING

by

Dazhou Liu

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2023

ABSTRACT

DETECTING THE ONION ROUTING TRAFFIC IN REAL-TIME BY USING REINFORCEMENT LEARNING

by Dazhou Liu

Anonymous networks have been popularly utilized to protect user anonymity and facilitate network security for a decade. However, such networks have been a platform for adversarial affairs and various network attacks including suspicious traffic generators. As a result, detecting anonymous network traffic is one critical task to defend a network against unpredictable attacks. Many new methods using machine learning and deep learning techniques have been proposed. However, many of them rely heavily on a vast amount of labeled data and have complicated architectures. Since network traffic always fluctuates under different network environments, those techniques may degrade in performance due to the network dynamics in real time.

Aiming to mitigate reliance on labeled data and simplify the structures of machine learning models, this study introduces a lightweight system to detect real-time anonymous network traffic leveraging the principles of reinforcement learning. Initially, the historical traces of anonymous traffic are analyzed to identify the crucial attributes that characterize anonymous and regular network traffic. Building on these important attributes, we design three components within the reinforcement learning framework: states, actions, and rewards. More importantly, decision-making thresholds that reflect the system's observations are set. Operating autonomously, the system employs these elements to discern network traffic categories in an unsupervised mode. Empirical results demonstrate that the system can identify patterns in anonymous traffic with an accuracy rate surpassing 80%.

ACKNOWLEDGEMENTS

Foremost, I want to express my sincere gratitude towards Dr. Younghee Park, for supplying the precious guidance along the path. I would also like to thank Dr. Stas Tiomkin and Dr. Mahima Agumbe Suresh for providing the invaluable insights.

Lastly, I'd express an appreciation to my family for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Tor - The Onion Router
IP - Internet Protocol
ML - Machine Learning
MDP - Markov Decision Process
TCP - Transmission Control Protocol
IDS - Intrusion Detection System
VPN - Virtual Private Network
DNS - Domain Name Service
API - Application Programming Interface

1 INTRODUCTION

1.1 General Trends of Anonymous Network

Anonymous networks play an important role in keeping a user's identity secret. Through these networks, both users and servers can communicate without disclosing identity-related information. Existing anonymous networks include the Onion Routing [1], Mix Network [2], and the Garlic Routing [3], all of which are designed to decouple the identities of the source and destination entities. Furthermore, encryption techniques have been prevalent and served as the foundation of secure communications [4]. When coupled with these encryption techniques, anonymous networks have advanced the level of privacy.

The demands for exploiting anonymous networks and visiting hidden network spaces surged. Approximately two million users accessed Tor and half a million users accessed hidden services via bridge, in the first quarter of 2020 [1]. Akoki et al. collected more than 2.3 million dark web pages and figured out that the dark web has become larger and more complex since June 2018 [5]. Investigations into the Tor network revealed that a broad range of illegal services and items, such as file sharing, ransomware panels, or counterfeits, have been circulating [6]. By abusing anonymous networks, the Tor network has been a practical tool for attackers to engage in criminal activities, such as malicious code distributions. All darknet-related traffic is assumed to be suspicious or malicious, due to the passive configuration of darknet [7]. Based on this assumption, the authors proposed to detect threats, such as Distributed Denial of Services (DDoS), by evaluating darknet traffic. These trends imply that anonymous traffic can be a sign of attacks.

To prevent penetrations, some systems are configured to block all anonymous traffic. Nonetheless, this action is detrimental to legitimate users who are privacy-conscious. Another countermeasure is to develop a blacklist of the IP addresses belonging to servers related to anonymous traffic [8]. When encountering suspicious traffic, the IP address of the incoming traffic is compared against each blacklisted IP address. However, this approach performs poorly when the blacklist is outdated. Furthermore, the extraction of blacklists by filtering technologies is expensive [9]. To improve the efficiency of detecting anonymous traffic, researchers inspect anonymous traffic based on statistics about packet headers and stream patterns [4].

Recently, machine learning and deep learning-based methods have been practical tools for analyzing anonymous traffic. Extensive studies have been on supervised learning [4], [10]–[12]. In supervised learning approaches, the objective is to learn the approximation of $P(y|x)$, given datasets in the form of $[x_i, y_i]_n$, where x_i is the feature set and y_i is the label set [13]. Within supervised learning algorithms, classifiers have been heavily employed to compute boundaries that divide samples into separate regions. In the Deep Learning field, classifiers are built with complicated structures and can process features in high dimensions. For instance, Convolutional Neural Networks are efficient in compressing and extracting information from two-dimensional feature spaces. Recurrent Neural Networks can learn features from sequential data and capture temporal characteristics.

Although classifiers are predominantly used, shortcomings exist when those models are deployed in real-time. For instance, in the realm of IDS, certain deep learning models show vulnerability to obfuscated features of network traffic and struggle to adapt to dynamic network patterns [14]. In terms of performance evaluation, some ML and Deep Learning-based models display a high False Positive Rate and lower F1-score in darknet traffic classification [12]. What's more, due to the limited availability of public datasets, researchers have difficulty accessing data about anonymous network traffic [15], [16].

Consequently, training detection models can be challenging. In light of these limitations, this paper proposes a reinforcement learning-based system for the detection of anonymous Tor traffic. By adopting an MDP framework, we defined the states, actions, and rewards to form a tuple. Based on the tuple, the proposed model takes actions, obtains reward values, and transits between states. Additionally, we analyze the distributions of the reward values and determine a threshold value for anonymous traffic and one for regular traffic. By comparing the reward value with the predetermined threshold, the agent decides whether the traffic is anonymous, regular, or ambiguous in real time.

1.2 Motivations

1. Many Tor traffic analysis methods rely on supervised learning. While these supervised models are highly efficient in classifying network traffic, their efficacy can be constrained by the scarcity of training data. In response, this study introduces a real-time unsupervised detection tool. The objective is to design an unsupervised detection tool with a small amount of data and yet exhibit an enhanced performance in circumstances with limited training data and labels.
2. The labeled darknet dataset, or CIC-Darknet-2020, provides historical traces related to anonymous and regular traffic [10]. Those traces were generated by online activities, including streaming, chatting, and browsing on platforms like Facebook

and Skype [4]. Rather than building intricate supervised models, we focus on analyzing and extracting the most relevant features of anonymous traffic.

3. Although network flows exhibit unpredictability, temporal dynamics can be observed between sequences of packets. Reinforcement learning methods have been effective in addressing sequential decision-making problems. To this end, a system adopting a deterministic MDP framework is developed for real-time monitoring and detection of anonymous traffic. This system is expected to interact with the network environment without the need for labeled training data.

The paper’s structure and content are outlined as follows. The related work section discusses the technical details of onion routing, introduces supervised learning-based detection methods for anonymous traffic, and explains applications of reinforcement learning frameworks to the network domains. Next, the methodology section describes the proposed reinforcement learning-based system. The experimentation and evaluations conducted are explained in the experiment section. Finally, the future work and conclusions summarize future scopes and the proposed system.

2 RELATED WORK

2.1 Overview of Tor and Darknet

2.1.1 Tor Network

The intention of the Tor network is to preserve anonymity. In turn, Onion Routing is designed as a distributed overlay network, aiming to anonymize TCP-related activities such as web browsing and messages sending [1]. Further, the Tor network consists of relays that are run by volunteers spanning the globe [17]. Each relay operates as a router that facilitates the reception of incoming traffic and the routing of outbound traffic to the intended destination.

To achieve anonymity, data are routed through a chain of no less than three relays, and each relay in a chain is only aware of its immediate predecessor and successor [1]. This design ensures that no intermediary can deduce the identity of the destination entity from the source entity, and vice versa.

To ensure the confidentiality of routed data, the Tor network enforces multi-layer cryptography. The layered cryptography resembles the structure of an onion. When a client, such as the Tor browser, transmits data through the Tor network, the local proxy encrypts the data layer by layer. When a relay in the Tor network receives data, it decrypts one of the layers.

As a result of the routing and encrypting mechanism, data routed through the Tor network attains a high level of anonymity and privacy.

2.1.2 The Darknet

In addition to communicating with entities like YouTube, the Tor network hosts servers that maintain web pages and offer services to Internet users. Those websites are known as Onion Services. The network of servers that support Onion Services can be considered as the darknet. As the darknet is restricted to be accessed through infrastructures like the Tor network, the Tor browser has become a prevalent tool for accessing the darknet. Differentiating Tor and darknet traffic is crucial for cybersecurity. By examining the open source codes of the Tor browser [17], four differences between darknet and Tor were identified. These characteristics include the locations of destination servers, the number of relays in a circuit, IP address, and DNS resolution.

For the locations of destination servers, the Tor network directs user data to servers outside the Tor network (e.g., YouTube) and within the Tor network (Darknet). The feature of circuit length describes the path to a public server or a darknet server. Specifically, the path to a public server consists of three Tor relays while the path to a darknet server contains six Tor relays. As for the IP address, the darknet servers conceal their IP address from the clients. As for DNS resolution, visiting servers outside the Tor network requires that the clients handle DNS resolution through the local proxy, or the exit relay resolves DNS on behalf of the client. However, communicating with darknet servers does not involve DNS resolution. Table 1 summarizes the four features.

Table 1
Summary of Differences between the Internet and Darknet

Server Location	**Circuit Length**	**Server IP Address**	**DNS Resolution**
Surface Webs	3 nodes	Disclosed to clients	By clients or exit nodes
Darknet	6 nodes	Hidden to clients	No Resolution

In summary, the Tor network has been built to refrain from traffic analysis. When the attackers abuse the Tor network, the security of information system resources is at risk. By examining the open source codes of the Tor browser, four features that characterize the darknet are unveiled. The relevant statistics provide more insights into detecting darknet traffic.

2.2 Detecting Tor traffic by Supervised Learning

As discussed in Section 2.1, reading or tracing anonymous Tor traffic is highly complicated. This aspect poses challenges to identifying Tor traffic from mixed network traffic. To address this issue, researchers extracted flow-level features and developed supervised learning-based approaches.

Lashkari et al. generated eight types of network traffic (e.g., browsing, chat, streaming) and captured Tor traffic between the client and entry node [11]. In their approach, eight categories of timing-related statistics were extracted. Those features include the amount of time between the arrival of two packets, the amount of time during which a flow remains active or idle, flow duration, and the number of bytes or packets in one second. Leveraging these features, Zero Rule, C4.5 Decision Tree, and K Nearest Neighbor were applied to classify network traffic into either Tor or normal class. Based on analysis, the C4.5 Decision Tree can detect 93.4% of Tor samples. Their findings indicate that timing-based features can be used to unveil Tor traffic patterns, even without accessing the packet contents.

Lashkari et al. merged the VPN-nonVPN dataset [4] and the Tor-nonTor dataset [11] to create a comprehensive darknet dataset [10]. Based on the dataset, they proposed a deep learning model in the form of a two-dimensional CNN. In the feature engineering process, 61 features were extracted based on descending feature importance scores generated by random forest. Subsequently, these features were harnessed to construct grayscale images with 64 pixels. Each pixel corresponds to one of the 61 feature values. By using convolution techniques, the deep learning model can classify Tor and non-Tor traffic with 95% test accuracy.

Mohanty et al. combined multiple classifiers by bootstrap aggregating to form an ensemble model [12]. In specific, the ensemble model comprises a base learner and a meta-learner. The first phase base learner is formed by the K Nearest Neighbor, Random Forest, and Decision Tree models. The succeeding meta-learner makes predictions by Logistic Regression. Evaluated on binary classification tasks, the stacking model is declared to achieve 98.89% accuracy.

To address the reliance of many classifiers on extensive labeled training data, an enhanced decision tree algorithm was introduced [18]. The authors identified four specific features that serve as the unique characteristics of Tor traffic. These four features are entropy related to packet length, frequency of appearance of packets with a length of 600 bytes, the number of packets with zero data in the first 10 packets, and the average time between the arrival of two packets. For the supervised model, a decision tree was constructed. Instead of adopting the splitting attributes in traditional C4.5 and ID3 decision trees, information gain is employed to select the most informative attributes.

Specifically, attributes with the highest information gain are chosen as the splitting attributes. During the testing phase, the authors collected network traffic and gathered 50,000 samples. Utilizing the four proposed features, the results indicate that the modified decision tree achieved an accuracy of up to 99% for detecting Tor traffic.

Due to privacy concerns and the limited availability of the dataset on anonymous traffic, researchers often gather private data or generate traffic within simulated environments [15]. In response to this challenge, the Anon17 dataset was introduced, logging features associated with Tor and other anonymous network traffic instances [15]. These features encompass packet header information, packet counts, and length in bytes of each flow. Leveraging the Anon17 dataset, the researchers probed the extent to which anonymous traffic can be identified by ML techniques [16]. The classification algorithms include Naive Bayes, Bayesian Networks, C4.5, and random forest. The results discern that classifiers can classify Tor and other types of network traffic instances with an accuracy closing to 100%.

In summary, the majority of supervised models designed for detecting anonymous Tor traffic rely on classification tasks. The efficacy of these models depends on the quality of data preprocessing, feature engineering, and the training process. When those models are trained with sufficient data and crafted features, they can detect anonymous traffic with an accuracy of over 90%. Conversely, the performance of supervised models may degrade due to inadequate data and non-relevant features. This aspect prompts the exploration of alternative decision-making paradigms.

2.3 Reinforcement Learning

Reinforcement learning involves the domain of sequential decision-making. In sequential decision-making problems, the goal includes learning what actions to execute at a state such that the expected returns are maximized. The mapping of a state to an action is referred to as a "policy". Unlike instructive approaches, reinforcement learning-based methods involve assessment of their behaviors. The evaluations of policies are quantified in terms of a value or the probability of taking an action at a state. Based on the evaluation strategy, reinforcement learning-based methods can be divided into two categories: value-based and policy-based learning.

2.3.1 Value-Based Learning

Upon following a policy, the quality of the policy is evaluated by a value metric. The value metric represents the expected cumulative return of a state or a state-action pair. In value-based learning, the value of a state or a state-action pair is encapsulated as a value function and iteratively updated. In policy iteration, the value of a state can be estimated by the Bellman equation [19]

$$v(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v(s')] \tag{1}$$

In Equation 1, each available action in a state is evaluated by summing the probability of transitioning to each subsequent state, multiplied by the immediate reward plus the discounted reward of the next state. The term $\sum_{s',r} p(s', r|s, a)[r + \gamma v(s')]$ implies that the expected cumulative return sums the values of all potential subsequent states upon executing an action. Additionally, the estimation requires knowledge about the transition probability to the next state, denoted as $p(s', r|s, a)$.

When the dynamic is not fully understood, model-free learning is an approach to solving MDP problems. Q-learning and State-Action-Reward-State-Action (SARSA) utilize Q-values to learn optimal policies that lead to the highest expected returns. Q learning methods leverage off-policy temporal difference (TD) error for learning. The temporal difference formula for updating a Q value is as follows [20]:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a Q(S\ ,a) - Q(S,A)] \tag{2}$$

The Q value of the action taken at the current state is updated by the TD error of

$$R + \gamma max_a Q(S' ,a) - Q(S,A).$$

Equations 1 and 2 are practical in scenarios where the state-action space is finite and discrete. To solve problems involving continuous state-action spaces, researchers studied substituting tables with neural networks. The reliance on crafted features and constraints posed by low dimensional state space was indicated by [21]. The authors subsequently introduced Deep Q Network to learn policies for controlling Atari 2600 games. Within their model, Q-values are parameterized as $Q(s, a; \theta_i)$, where θ_i is the weight set of a convolutional neural network. The temporal difference formula was harnessed to update the parameters of the neural network. To alleviate data correlations and stabilize the learning process, the authors implemented experience replay and target network. When evaluating the Deep Q Network using Atari games, their framework outperforms human players and baselines in 29 games in terms of game scores.

2.3.2 Policy-based Learning

There are situations in which the estimation of values is intractable. One solution is to parameterize and optimize the policies directly. Such policy optimization approaches include policy gradient [22] and Trust Region Methods [23]. In policy gradient, the parameterized policy takes a state vector as the input and outputs the probability distribution of actions. Next, the gradient of the policy is calculated and the policy is updated by gradient ascent.

Trust region policy optimization (TRPO) stabilizes the policy updates by constraining the magnitude of each update [24]. To make further improvements based on vanilla policy gradient, Q learning, and TRPO, Proximal Policy Optimization (PPO) was proposed [24]. In PPO, the data efficiency and reliability of TRPO are further enhanced by clipping the surrogate objective. At the one-million step benchmark in continuous control tasks, the results exhibit that PPO outperforms other implementations with the highest episode rewards. In actor-critic architecture, the actor neural network acts as the policy function and the critic neural network behaves as a value function that estimates the actor.

2.3.3 Applications in the Network Environment

Reinforcement learning frameworks have a wide range of applications for optimizing network topologies and developing counter-attack strategies.

A reinforcement learning-based approach is developed to make optimal routing decisions while satisfying security requirements [25]. In the proposed method, each state is represented as a switch on the data plane of the Software Defined Network. In addition, security devices are deployed on some of the switches. The goal of the agent is to traverse from the source switch to the destination switch while avoiding paths that have high latency, jitter, and packet drop rates. The proposed method utilizes a Q table and defines the reward function as the weighted sum of delay, jitter, traffic rate, and packet loss.

Compared to existing link stability-based Q-routing, the results state that the delay time is reduced, regardless of the number of security constraints. A Deep Q Learning technique is applied to counter jamming attacks in cognitive radio networks (CRN) [26]. In a CRN system, the participants include primary users (PU), secondary users (SU), and jammers. In the proposed method, each state is represented as the appearance of PUs and signal-to-interference-plus-noise ratio (SINR). The agent is an SU and its action is choosing to either leave the jamming area or defeat the jammer.

Addressing the limitations of Q learning to large state space, a deep convolutional neural network is leveraged to approximate the Q values. The result shows that the proposed method has a faster convergence time and achieved higher SINR, compared with the naive Q learning-based method.

A Q learning paradigm is combined with a classifier to develop an IDS [27]. In addition to the classifier agent, the environment serves as the second agent. The state of the classifier agent is interpreted by samples in the NSL-KDD dataset, and the action of the classifier agent is to predict labels. Simultaneously, the action of the environment is to select a state for the classifier, such that the prediction accuracy of the classifier agent is reduced. In this adversarial mode, the proposed method has the highest accuracy and lowest prediction time among other supervised baselines. Whereas, the performance of the proposed method is lower than that of other reinforcement learning algorithms like the Asynchronous Advantage Actor Critic (A3C).

According to the applications of reinforcement learning, we define the actions, states, and rewards, such that these definitions fit the characteristics of a finite and discrete state-action space. Based on these definitions, this study develops a value-based approach to detect Tor traffic.

3 METHODOLOGY

This section explains the network environment in which the proposed detection system is to be deployed and the adapted MDP framework to detect anonymous traffic.

3.1 Network Environment

3.1.1 Information to be Extracted

In a network environment, anonymous traffic is expected to be present at any time. Meanwhile, real-time detection of anonymous network traffic necessitates a comprehensive understanding of traffic patterns. Rather than conducting deep packet inspections on individual packets, the system extracts pattern information from a sequence of packets. Each sequence of packets, akin to a flow, is transmitted between the same source and destination IP addresses, on the same source and destination port [4]. A flow is transmitted in the forward direction (FWD) if the source sends packets to the destination [4]. Otherwise, the direction is backward (BWD).

Within this framework, the flow-level information includes features such as flow length in bytes, packet counts, duration of a flow, and the time elapsed between the arrival of consecutive packets in the FWD or BWD. For instance, the average length of each packet within a flow can be calculated by dividing the total number of bytes by the total number of packets. Given that no training data is available in real-time, the system computes those statistics upon capturing a flow.

3.1.2 Environment Descriptions

The environment is a dynamic network space constantly receiving traffic. Due to constraints in dimensionality, Fig. 1 visualizes the environment in two dimensions and displays statistics related to the number of packets. In Fig. 1, each data point corresponds to a captured flow. The x-axis and y-axis denote the elapsed time since the system's initiation and the number of packets in a flow. For instance, at time 0 the system captured a flow comprising 23 thousand packets in the FWD direction. At the 80 milliseconds (ms) mark, the system captured a flow containing 110 thousand packets in the BWD direction.

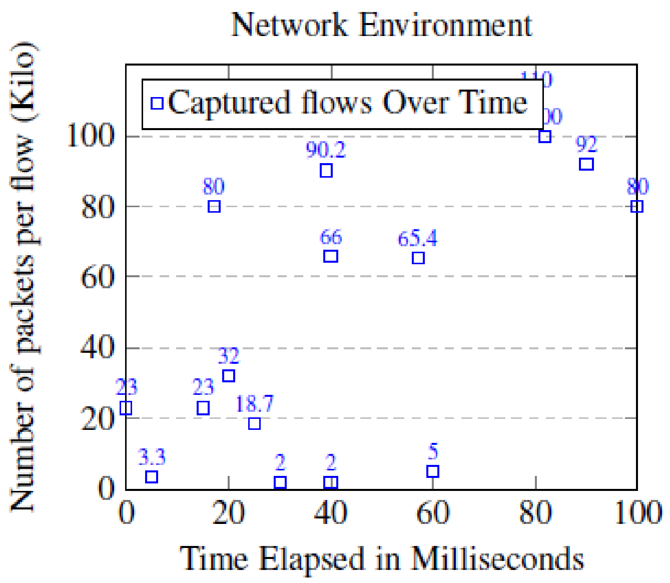

Fig. 1. The network environment with mixed anonymous and regular network traffic.

Upon initiation, the monitoring procedure is as follows. In the beginning stage, the system keeps listening on a port and awaits the arrival of network flows. When a flow is captured, which could occur at 39ms, the system is unable to discern whether this flow is anonymous. Next, the agent extracts information from each flow's predefined feature sets. For instance, it quantifies the packet counts and flow length in bytes, as described in subsection 3.1.1. After obtaining those flow statistics, the environment dispatches a reward signal to the agent. According to the reward value, the system decides whether the captured flow is ambiguous. After each decision, the agent concludes its evaluation of the current flow and awaits the analysis of the next arriving flow, which could occur at 40ms. As the proposed system aims to operate in real-time, it is expected to decide the flow status with shorter than the time elapsed between consecutive flows.

3.2 Proposed Detection Model

MDP is a standard framework for addressing problems related to sequential decision-making [28]. A quadruple (S, A, r, p) can be used to design an MDP environment, where S is a finite set of states, A is a finite set of actions, p is a transition probability, and r corresponds to the immediate reward received after executing an action [28].

Aligning with the tuple, the detection model is designed to emulate the MDP model, as depicted in Fig. 2. In Fig. 2, the process involves selecting a set of features (e.g. the average length of packets in a flow and idle time) by the system, according to the current policy. Next, following the execution of action A_t, the environment sends the reward to the agent, leading to a transition to another state. This design addresses the adaption of the MDP paradigm to the network environment and the interplay between the agent and the environment.

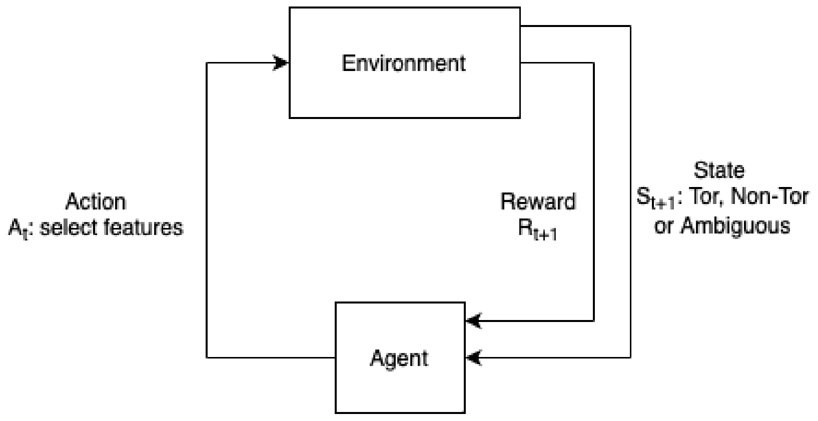

Fig. 2. Scheme description.

Based on the CIC-Dakrnet2020 dataset [10], the list below summarizes each element in the quadruple.

1. **State S:** The state space contains three states: Tor, non-Tor, and ambiguous. The state space is interpreted as $S = \{s1, s2, s3\}$. In S, s1 represents an ambiguous state, s2 represents a Tor state, and s3 represents a non-Tor state. The initial state of the agent is ambiguous, meaning that the agent has no clue about the behavior of the detected traffic. As such, the goal of the agent is to identify an answer so as to leave the ambiguous state. Accordingly, both the Tor and non-Tor states are considered as the goal states. When reaching the Tor state, the agent assumes that the current traffic is generated by the Tor network. When transiting to the non-Tor state, the current traffic flow is recognized as non-Tor. In this design, the conventional labels used for classification tasks are replaced with these model states, allowing the system to process the observed network traffic differently.
2. **Action A:** Rather than assigning labels, the action of the agent is to select features. The action space is defined as $A = \{a1, a2, a3, ..., an\}$, where $ai \in A$ and $ai = \{f1, f2, f3, ..., ft\}$. The subscript n represents the number of actions, and t is the number of features selected by each action. To reduce the complexity and address the feature importance, the feature selection techniques provided by the scikit-learn library [29] are used to select features. To utilize the non-linear attribute of random forest and reduce overfitting, those selection techniques are chosen as Recursive Feature Elimination (RFE), Recursive Feature Elimination with Cross Validation (RFE), Random Forest Classifier, and SelectKBest. Table 2 summarizes the action space.

Based on this design, the action space size is four. The first, second, third, and fourth action is selecting features ranked by RFE, RFECV, Random Forest Classifier, and SelectKBest, respectively. The number of features along with the selected features by each action will be discussed in the experiment section.

Table 2
Action Space

Number	Action
1	Select features ranked by RFE
2	Select features ranked by RFECV
3	Select features ranked by Random Forest Classifier
4	Select features ranked by SelectKBest

3. Heuristic **Reward Function:** As an indicator of the consequences of the agent's action, the reward function is designed heuristically. The immediate reward is calculated using a specific formula that involves weighted feature values and a hyperbolic tangent function. More specifically, it is calculated as taking the linear sum of weighted feature values as input to the hyperbolic tangent function. This design of the reward function bounds the values of the reward within the range of -1 and 1. Besides, it can potentially separate Tor and non-Tor samples according to the thresholds around 0. The reward function formula is as follows:

$$r(s,a) = \tanh(w_1 \times f_1 + w_2 \times f_2 + w_3 \times f_3 + ... + w_t \times f_t) \quad (3)$$

The Hyperbolic Tangent function, tanh(x), is in the form of:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4)$$

where e is the natural exponent and x is the linear sum of the weighted feature values, as shown in Formula 3.

4. **Environment Model:** In some MDP problems, the model of the environment is the transition probability, or *P(s_next|s, a)*. Specifically, the model is the conditional probability of transiting to the next state, given an action *a* at the current state. Depending on knowledge about the environment model, the interactions with the environment can be model-based or model-free. In scenarios such as balancing a pendulum or humanoid, the environment model can be approximated by principles such as kinematics. However, an explicit dynamic among the recorded network flows in CIC-Darknet2020 has not been identified. Thus, the agent is expected to make decisions and interact with the environment without being aware of the environment model.

3.3 Threshold Setting

The introduction of thresholds is a strategy aimed at reducing the reliance on labeled data, which helps the detection system make a decision. Two thresholds are established: one for non-Tor traffic and one for Tor traffic. Segmented by the Tor and Non-Tor thresholds, three intervals are formed: the interval left and right of the non_Tor and Tor thresholds along with the region between the non-Tor and Tor thresholds. Fig. 3 describes the three intervals formed by the two thresholds.

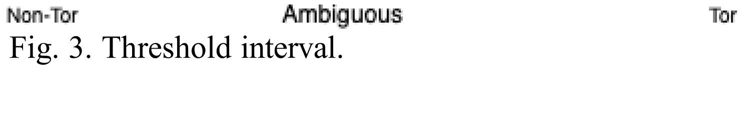

Fig. 3. Threshold interval.

When the agent chooses an action in response to an observed network flow, the environment assigns the immediate reward and compares it with the predefined thresholds. If the reward value is lower than the non-Tor threshold (left of the non-Tor threshold), the agent classifies the current traffic as non-Tor traffic with certainty. If the reward value is higher than the Tor threshold (right of the Tor threshold), the agent confidently classifies the current traffic as Tor traffic. In the third scenario, the reward value falls between the two thresholds and the agent cannot make a decision. This scenario indicates ambiguity regarding the type of current traffic. It is worth noting that ambiguous data prompt the need for further processing. Song et al. state that ambiguous samples impact accuracy negatively and are non-trivial when the number of data points is huge [30].

By setting the thresholds, Tor and non-Tor flows can be distinguished in an unsupervised mode, and the labels are substituted with threshold values.

3.4 State Transitions

3.4.1 Detection Model Illustrations

Fig. 4 depicts the three states and four actions of the detection model. The nodes labeled *Ambiguous*, *Tor*, and *non-Tor* represent the three states. The edges illustrate the transitions between states after applying an action. In Fig. 4, $a1$ corresponds to the first action in the action space, which is selecting features ranked by RFE, as shown in Table 2. The actions $a2$, $a3$, and $a4$ represent the second, third, and fourth actions in the action space. Upon each action, the reward, r, is calculated according to Equation 3. Based on this three-state graph, the agent can reach the other two states from arbitrary starting states, covering all three transition possibilities.

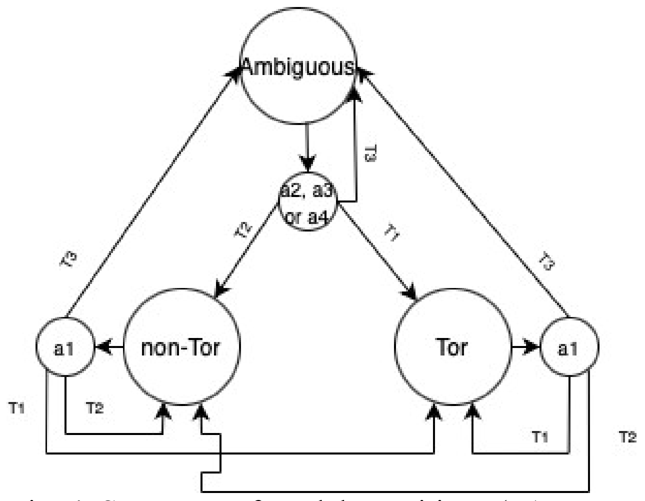

Fig. 4. Summary of model transitions (T1: move to Tor state (r ≥ Tor threshold), T2: move to non-Tor state (r ≤ non_Tor threshold), T3: move to ambiguous state (non_Tor threshold < r < Tor threshold)).

At the Tor or non-Tor state, the agent has a single option of applying $a1$. As depicted in Fig. 4, taking action $a1$ results in moving to the other two states or staying in the same state. For instance, in the non-Tor state, there are three transition possibilities. First, the agent stays in the non-Tor state if the reward value received is lower than the non_Tor threshold. In the second scenario, the agent moves to the Tor state from the non-Tor state when it receives a reward value higher than the Tor threshold, as mentioned in Fig. 3. In the third scenario, the agent moves to the ambiguous state if the reward value is less than the Tor threshold and higher than the non_Tor threshold.

In particular, Fig. 5 illustrates the agent's behavior in the ambiguous state. The ambiguous state indicates that action $a1$ is not informative to make a decision. Once transiting to the ambiguous state from the Tor or non-Tor state, the agent applies $a2$, $a3$, and

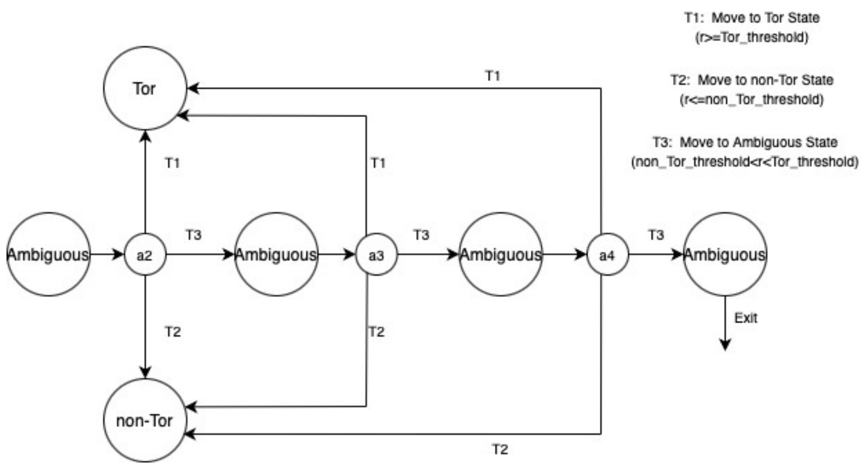

Fig. 5. Transitions at the ambiguous state.

*a*4 sequentially until one of these actions leads the agent to transit back to either the Tor or non-Tor state. However, if the agent ends up in the ambiguous state after applying the action *a*4, the current flow is labeled as anonymous and malicious.

Overall, an action that results in the agent transiting to the Tor or non-Tor state or staying in the ambiguous state after applying *a*4 concludes the current flow.

3.4.2 Detection Algorithm

Algorithm 1 below describes the transitions between states according to actions and rewards. At either the Tor or Non-Tor state, the agent labels the current flow and moves to the next flow. On the other hand, if the agent ends up reaching the ambiguous state, it selects the other three actions in the action space. Once an action results in a reward value greater than the Tor threshold or less than the non-Tor threshold, the agent concludes the current flow and moves to the next flow. If the agent remains in the ambiguous state even after applying four actions, the traffic statistic is stored in a buffer, and the agent skips to the next flow. Elements in the buffer require further analysis depending on the policies of the corresponding IDS.

In the implementation phase, the flows are processed in batches to reduce the time overhead.

Algorithm 1 Transitions Between States

```
1:  procedure TRANSITION(threshold_Tor, threshold_nonTor, feature_set, batch_size)    ▷ Function
2:      for i in range(0, len(feature_set), batch_size) do
        // Create batches
3:          batch_features = feature_set[i : i+batch_size]
        // Calculate reward value of each sample in the batch
4:          r_batch = first_action(batch_features)
5:          for j in range(len(batch_features), batch_size) do
6:              if r[j] >= threshold_tor then                          ▷ Classified as Tor
7:                  State ← Tor
8:              else if r[j] <= threshold_non_tor then                 ▷ Classified as non_Tor
9:                  State ← Non_Tor
10:             else                                                   ▷ Classified as Ambiguous
11:                 r_batch = next_action(batch_features)
12:                 while r[j] > threshold_non_tor and r[j] < threshold_tor do
13:                     if no actions remaining in action space then
14:                         Label (Anonymous && Malicious)             ▷ Further Processing
15:                     break
16:                     State ← Ambiguous
17:                     Select next action
```

4 EXPERIMENTS AND EVALUATIONS

This section delineates the implementation and testing of the proposed framework. At first, the dataset was cleaned and preprocessed. Second, the feature sets to be used for constructing the action space were selected. Third, each reward function corresponding to the respective action is defined. Lastly, the performance of the framework is gauged based on accuracy.

4.1 Dataset Processing

The CIC-Darknet2020 dataset is an archived dataset with labels. It summarizes the features of bidirectional traffic flows generated by the CICFlowMeter [10]. As this dataset is readily available, our model was developed and tested based on those historical flows. As the Tor browser is commonly used and anonymous traffic is often encrypted, we encompass Tor and VPN traffic samples within the realm of anonymous traffic.

The features can be divided into two categories: time-based features and packet-based features. Time-based features include packet inter-arrival time, idle time, and byte per second. Packet-based features encompass statistics about packet size and counts of flags per flow.

There are four types of labels: non-Tor, non-VPN, Tor, and VPN. Concerning that Tor traffic is encrypted, samples labeled as VPN are considered Tor samples. On one hand, anonymous Tor traffic is regarded as an anomaly. On the other hand, the non-Tor and non-VPN labels are merged into the non-Tor type. After this amalgamation, two types of labels remain in the dataset, which are non-Tor and Tor. Table 3 summarizes the raw dataset.

Table 3
Raw Dataset

Total Samples	Total Features	Non-Tor Samples	Tor Samples
141530	83	117219	24311

To elaborate, the number of recorded flows is 141,530. The number of features is 83. The number of flows labeled as non-Tor and Tor is 117,219 and 24,311, respectively.

4.1.1 Cleaning and Preprocessing

The raw dataset was imported and converted to a *pandas dataframe*. Next, the data was cleaned. In the cleaning phase, rows that contain invalid values, such as infinity and “Not a Number” are dropped. Then the columns whose entries are all equal to the same value were discarded. As a result, the number of features is reduced from 83 to 62 and the number of samples is decreased from 141,530 to 141,483. To further simplify the feature set, the Feature Selection with Variance Thresholding (VarianceThreshold) technique in Scikit-Learn library (abbreviated as sklearn) [29] is applied. The VarianceThreshold technique removes features with a variance below a specified threshold. In the experiment, the threshold is set to 30%. After running VarianceThreshold, four features were discarded, resulting in 58 features.

The raw dataset is skewed. As shown in Table 3, Non-Tor samples are abundant. On the other hand, the number of Tor samples is 24,311 and only accounts for 17.18% of the 141,483 labels. To mitigate the bias, the samples indexed at 0-69999 were eliminated. Next, we generated synthetic data by duplicating 20,000 samples labeled as either Tor or VPN.

After processing, the finalized dataset contains 91,483 samples and 58 features. As Table 4 displays, the final dataset contains 47,172 Tor samples, accounting for 48.4% of the 91,483 samples. The number of features is 58.

Table 4
Final Dataset

Total Samples	Total Features	Non-Tor Samples	Tor Samples
91483	58	47172	44311

Lastly, the order of samples is shuffled to reduce overfitting and bias.

4.1.2 Label Encoding

Since each label is categorical and stores a string, each label is replaced by an integer of 0 or 1 to follow the Bernoulli Equation. Bernoulli Equation is defined as $P(X = 0) = 1 - p$ and $P(X = 1) = p$, where X is a random variable and p is the probability of X being equal to 1. Since anonymous traffic is defined as an anomaly, Tor labels are replaced with integer 1 (Positive Class). The labels of non-Tor are replaced with 0 (Negative Class). This step is achieved through the Label Encoding method of sklearn.

To apply the hyperbolic tangent function to the classification model, the labels belonging to the Positive Class are converted to 1 while the labels of the Negative Class are converted to -1. The formula is as follows: $label\_encoding = 2 \times label\_encoding - 1$.

4.2 Feature Selection Experimentation

4.2.1 Constructing the Action Space

The process of finalizing the action space involves determining the number of features selected by each action and the number of actions in the action space.

In terms of the number of features selected by each action, this study chose to set this number within the range of 10 to 15 features. This choice was based on prior experience and information from earlier research on the classification of Tor traffic. An alternative perspective is to provide each classifier with as much information as possible, implying that all 58 features in the dataset would be used. Nevertheless, this setting increases the data volume that needs to be processed.

To determine the appropriate number of features to be selected by each action, the study employs the accuracy metric to evaluate the performance of the Random Forest Classifier. The evaluation was conducted while increasing the number of features in step of 15. Additionally, these features were included following the importance score generated by the Random Forest model after fitting. Fig. 6 delineates the accuracy curve. Initially, the accuracy increased from 0.9746 by using 15 features to 0.9774 by using 30 features. Nonetheless, the accuracy ceases to increase when more than 30 features are used. This observation indicates that increasing the number of features within a certain range contributes to improving the accuracy. However, an exceeding number of features do not contribute to improving the performance. According to this phenomenon, the number of features selected by each action is finalized as 15.

Regarding the number of actions within the action space, we considered two distinct approaches: randomly selecting 15 features from 58 features and selecting features with the top importance scores. When randomly selecting features, the number of combinations by choosing 15 features from 58 features is estimated to be 2.97×10^{13}. A dimension of the action space on the scale of a trillion is impractical to be solved in discrete cases. The second

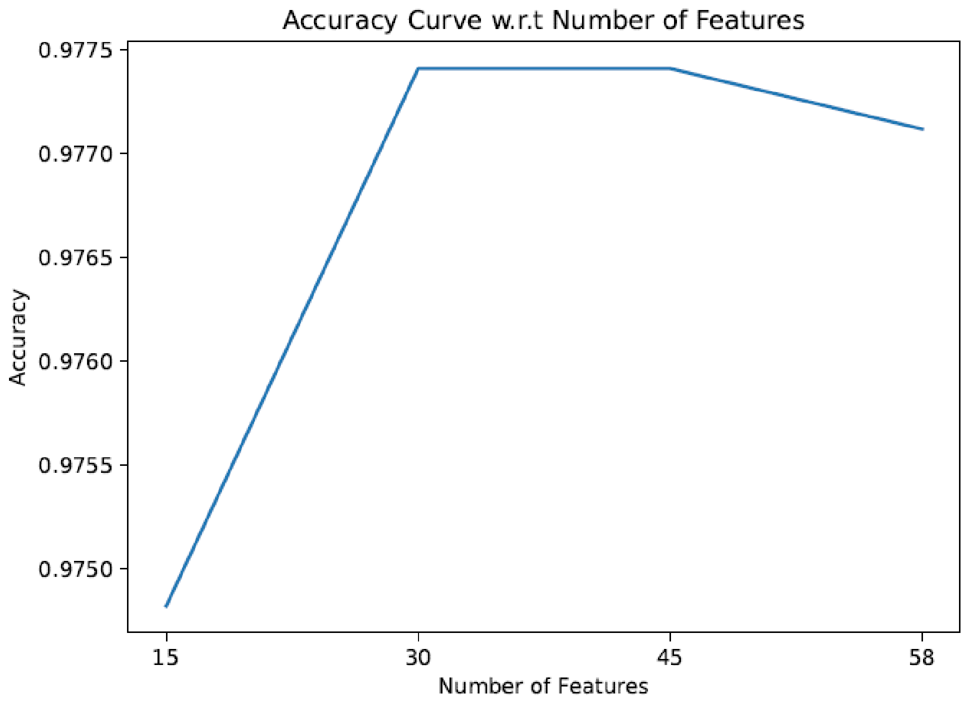

Fig. 6. Accuracy trend of random forest classifier regarding number of features.

approach aims to select features based on the importance scores. For this purpose, this study employed Scikit-Learn APIs of feature selection techniques, including Recursive Feature Elimination (RFE), Recursive Feature Elimination with cross-validation (RFECV), SelectKBest, and Random Forest Classifier. Each of these techniques corresponds to a specific action, effectively reducing the size of the action space to four.

This configuration concludes the action space construction with four actions and 15 features to be selected by each action.

4.2.2 Hyper-Parameter Setting

This subsection explains the hyper-parameter settings for the feature selection techniques.

Given an importance score-based estimator, RFE removes the least important features recursively until the specified number of features is reached. In the experiment, the specified number of features is 15. Cross Validation can be combined with Recursive Feature Elimination as Recursive feature Elimination with Cross Validation. The number of cross-validation in RFECV is set to three for a faster convergence. The SelectKBest tool selects features with the highest k importance scores ranked by a score function. In the experiment, the score function utilized is Mutual Information for discrete class. As for the Random Forest Classifier, it ranks features based on importance scores that are calculated by using entropy or the Gini coefficient.

Since the external estimator used for RFE and RFECV is the random forest classifier, the grid search technique [31] is leveraged to determine the most suitable hyper-parameters. The criteria to be searched are the number of estimators, the function for measuring the split quality, the maximum depth, the minimum samples to split, and the minimum samples required for being a leaf node. Upon multiple running of grid search based on all 58 features, we decided to adopt the combination of hyper-parameters as displayed in Table 5. As depicted in Table 5, the bootstrap is set to True, consistent with the default setting. The criterion for evaluating the quality of node splitting is the Gini index. Each leaf node is configured to contain at least one sample, while a minimum of ten samples are required to split a non-leaf node. Lastly, the number of individual trees in the random forest ensemble is set to 125.

After grid searching, the random forest classifier is fitted with all 58 features.

4.2.3 Feature Ranking Results

The resulting features for each action are summarized in Table 6 and Table 7, which form the first column of Table 2.

Table 5
Random Forest Classifier Parameters

Parameters	**Value**
bootstrap	True
criterion	gini
min_samples_leaf	1
min_samples_split	10
n_estimators	125

Table 6
Feature Rankings by RFE and RFECV

Rank	**RFE**	**RFECV**
1	Bwd Packet Length Min	Flow Duration
2	Bwd Packet Length Mean	Total Length of Fwd Packet
3	Flow Packets/s	Total Length of Bwd Packet
4	Flow IAT Mean	Fwd Packet Length Min
5	Flow IAT Max	Bwd Packet Length Max
6	Flow IAT Min	Bwd Packet Length Min
7	Fwd Header Length	Bwd Packet Length Mean
8	Bwd Packets/s	Flow Bytes/s
9	Bwd Segment Size Avg	Flow Packets/s
10	Subflow Bwd Bytes	Flow IAT Mean
11	FWD Init Win Bytes	Flow IAT Std
12	Fwd Seg Size Min	Flow IAT Max
13	Idle Mean	Flow IAT Min
14	Idle Max	Fwd IAT Total
15	Idle Min	Fwd IAT Mean

Table 7
Feature Rankings by SelectKBest and RF

Rank	**SelectKBest**	**RF**
1	Flow IAT Max	Flow IAT Min
2	Flow Duration	Idle Max
3	Flow IAT Mean	Bwd Packet Length Min
4	Flow Packets/s	Fwd Seg Size Min
5	Fwd Packets/s	Flow IAT Mean
6	Bwd Packets/s	Bwd Segment Size Avg
7	Flow IAT Min	Subflow Bwd Bytes
8	Flow Bytes/s	Flow Packets/s
9	Average Packet Size	Bwd Packet Length Mean
10	Packet Length Mean	Idle Mean
11	Packet Length Max	Fwd Header Length
12	Packet Length Std	Flow IAT Max
13	Packet Length Variance	Bwd Packets/s
14	Bwd Segment Size Avg	Flow Bytes/s
15	Bwd Packet Length Mean	Fwd Packets/s

For RFE, the most important feature is identified as *Bwd Packet Length Min* while the feature with the 15th rank is *Idle Min*. Similarly, for RFECV, the most important feature is *Flow Duration* while the feature ranked as the 15th is *Flow IAT Mean*. Furthermore, there are common features that are selected by RFE, RFECV, SelectKBest, and RF. Features such as *Bwd Packet Length Min*, *Flow Packets/s*, and *Flow IAT Max* are consistently determined as informative for distinguishing between Tor and non-Tor traffic.

To verify the consistency of the feature selection results across multiple runs of RFE, RFECV, SelectKBest, and RF, the feature ranking of each run was compared. It is discovered that approximately 13 out of 15 features were consistently identified in each run, although their ranking positions may vary. As a result, the features from the last run are adopted as the final selection.

4.3 Reward Function Experimentation

Initially, the efforts were on computing the weights of a linear equation such that the aggregate of weighted features separates Tor and non-Tor samples around a zero threshold. This approach resembles linear regression. Nonetheless, the empirical findings imply that the relationship between feature values and the labels is intricate. To address this issue, non-linearity is introduced to facilitate separating Tor and non-Tor samples around the zero threshold. Based on the experiment, the hyperbolic tangent function is adopted. The subsections below discuss the outcomes.

4.3.1 Hyperbolic Tangent-Based Reward Function

A single-layer neural network with no hidden layer is harnessed to implement the reward function mentioned in Equation (3). Specifically, the weights of the reward function are derived from the trained neural network. The activation function at the output layer serves as the function for transformation. In each instance of the single-layer neural network, the input layer contains 15 neurons while the output layer has one neuron. The hyperbolic tangent function is employed at the output layer to map the linearly weighted summation of features onto a single value.

In preparation for training, the features of each sample undergo standardization to reach unit variance. The reason for applying standardization is that many feature values have a vast range. For instance, the *idle max* feature has a maximum value of 1.44×10^{15}, whereas the *Bwd Packet Length Min* feature has a maximum value of 1,350. The class of *StandardScaler* class of sklearn [29] is leveraged to bring all feature values to the same scale. During the training phase, each neural network is trained in a supervised manner by using the PyTorch library. Stochastic gradient descent (SGD) with a learning rate of 0.001 is used as the optimizer. It is observed that the loss stops decreasing after 30 epochs.

Consequently, each neural network is trained with 30 epochs. After 30 epochs, it was found that the weights in each instance of the neural network fluctuated, even when the same feature set was applied. To determine the influence on the detection accuracy, we tested the performance of each trained neural network. The result is that varying weights after training do not reduce the accuracy, provided that the feature sets are used consistently across training and testing. As a result, the weights produced by the last run are finalized as the weights of each reward function. The results of the reward functions are recorded in Tables 8 and 9.

The results showcase that the ranking of weights does not align with the ranking of feature importance generated by the sklearn feature selection techniques. For instance, within

Table 8
Weights Results by RFE and RFECV

Rank	W1	a1 (RFE)	W2	a2 (RFECV)
1	1.2904	Bwd Packet Length Min	0.2689	Flow Duration
2	-0.0694	Bwd Packet Length Mean	0.1872	Total Length of Fwd Packet
3	-0.0365	Flow Packets/s	-0.1440	Total Length of Bwd Packet
4	-0.0984	Flow IAT Mean	0.0584	Fwd Packet Length Min
5	-0.0583	Flow IAT Max	-0.1191	Bwd Packet Length Max
6	-0.3272	Flow IAT Min	0.7849	Bwd Packet Length Min
7	0.1684	Fwd Header Length	0.1817	Bwd Packet Length Mean
8	-0.2140	Bwd Packets/s	-0.5075	Flow Bytes/s
9	-0.0785	Bwd Segment Size Avg	-0.0699	Flow Packets/s
10	0.1425	Subflow Bwd Bytes	-0.1312	Flow IAT Mean
11	0.0707	FWD Init Win Bytes	-0.2003	Flow IAT Std
12	0.4345	Fwd Seg Size Min	-0.1743	Flow IAT Max
13	0.0277	Idle Mean	-0.2615	Flow IAT Min
14	-0.2260	Idle Max	-0.0326	Fwd IAT Total
15	0.1763	Idle Min	0.2427	Fwd IAT Mean

Table 9
Weights Results by SelectKBest and RF

Rank	W3	a3 (SelectKBest)	W4	a4 (RF)
1	0.2543	Flow IAT Max	-0.0083	Flow IAT Min
2	0.6159	Flow Duration	-0.2216	Idle Max
3	-0.3229	Flow IAT Mean	0.1781	Bwd Packet Length Min
4	-0.0131	Flow Packets/s	0.1014	Fwd Seg Size Min
5	-0.2443	Fwd Packets/s	-0.1772	Flow IAT Mean
6	-0.1869	Bwd Packets/s	-0.1262	Bwd Segment Size Avg
7	-0.2665	Flow IAT Min	-0.0275	Subflow Bwd Bytes
8	0.1950	Flow Bytes/s	-0.0723	Flow Packets/s
9	-0.2263	Average Packet Size	-0.0743	Bwd Packet Length Mean
10	-0.4022	Packet Length Mean	-0.1299	Idle Mean
11	-0.2983	Packet Length Max	0.1927	Fwd Header Length
12	-0.1267	Packet Length Std	0.0887	Flow IAT Max
13	-0.3179	Packet Length Variance	0.2001	Bwd Packets/s
14	0.4655	Bwd Segment Size Avg	-0.1554	Flow Bytes/s
15	0.6879	Bwd Packet Length Mean	-0.2383	Fwd Packets/s

the feature set selected via RFE, *Idle Min* has the lowest ranking along with a third-highest weight of 0.1763. Meanwhile, *Flow IAT Max* is ranked at the top position by SelectKBest. However, it has the third highest weight. This discrepancy suggests that the weights in a neural network and the feature importance are different metrics in evaluating the contributions of features to predictive performance.

Based on Tables 8 and 9, the reward function corresponding to each action is represented as R(s, a_t) = tanh$(W^T \times a_t)$. Hence, the reward of $a1$ is calculated *as* $R(s, a_1)$ = tanh$(W^T \times a_1)$, where the weight set is transposed and multiplied with the features selected by action $a1$.

4.4 Threshold Probing

Setting appropriate thresholds involves adjusting the level of sensitivity to Tor traffic. Since the target thresholds are expected to be around 0, the probing process initiates at 0. The Tor and non_Tor thresholds were tested in the range of -1 to 1, and the incremental step is 0.05. Fig. 7 depicts the trends.

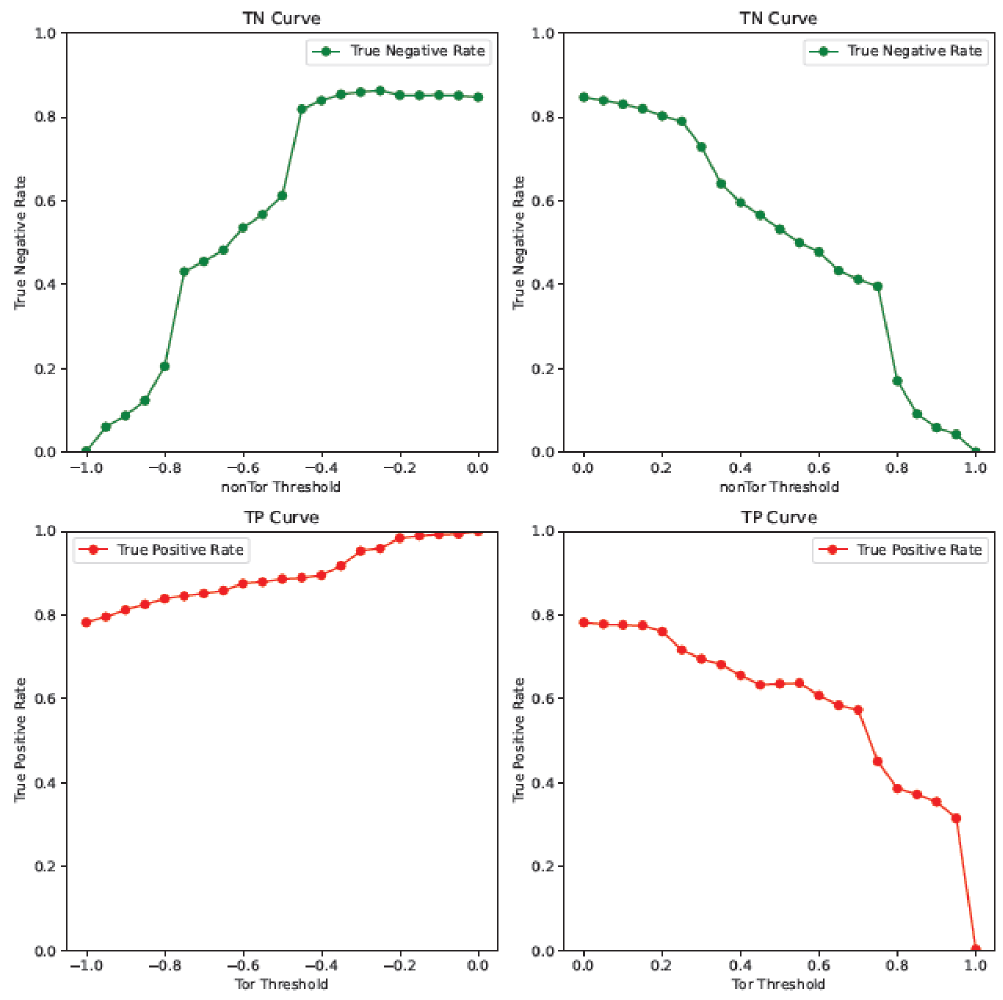

Fig. 7. True positive and true negative curve.

Based on the variations of the true negative and true positive rates, the true positive and true negative rates increase when increasing the Tor and non_Tor thresholds from -1 to 0. However, the true negative and true positive rates decrease as the thresholds keep increasing after 0. To maximize the true positive and true negative rate, the Tor and non_Tor thresholds are finalized as -0.1 and -0.3 respectively.

4.5 Results of Model Testing

The testing procedure of the unsupervised model is conducted in a supervised manner. Specifically, every time a reward value is obtained, the reward value is compared with the thresholds. The state to which the agent transits is compared with the known label for the current sample. If the state and label match, the sample is correctly identified. Conversely, if the state and label do not match, the samples are misclassified. In the event of entering the ambiguous state, the agent continues to apply the subsequent actions until it receives a reward value of either above the Tor threshold or below the non_Tor threshold. If all four actions are applied but the agent remains in the ambiguous state, the agent classifies the sample as ambiguous. Depending on the administrative configurations, these samples can be flagged as malicious and further analyzed by other IDS tools.

Fig. 8 presents a breakdown of true positives, true negatives, false positives, and false negatives. Based on the confusion matrix, 36,825 Tor samples are correctly identified, while 8,886 Tor samples are misclassified as non-Tor. As for non-Tor samples, 38,286 are correctly classified, while 7,486 samples are misclassified as Tor samples, resulting in false positives. Using the accuracy formula, or (*true positive* + *true negative*)/(*true positive* + *true negative* + *false positive* + *false negative*), the accuracy rate is calculated as 82%. Additionally, the

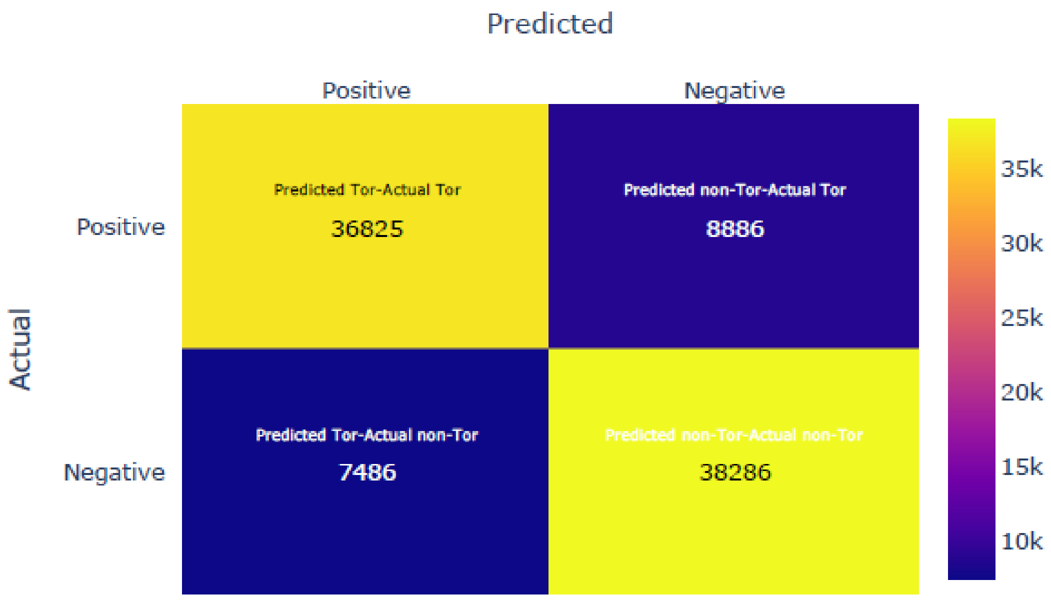

Fig. 8. Confusion matrix of detection accuracy. (Note that Tor threshold=-0.1 and Non-Tor threshold=-0.3).

performance was evaluated using the metrics of recall and precision. The result is that the precision is 0.83 and the recall is 0.81. Nonetheless, the cumulative count of samples is 91,483, which is equivalent to the size of the dataset, indicating that no sample is labeled ambiguous by the detection model. This outcome demonstrates that the features selected by all four actions can effectively make the model deduce the type of each traffic sample.

Fig. 9 illuminates the reward value distribution with respect to the thresholds. The Tor Threshold is set at -0.1, the non-Tor threshold is set at -0.3. The dashed black horizontal lines denote the thresholds.

Within Fig. 9, the figure labeled as Tor distribution represents instances where Tor samples are correctly classified. Those correctly classified Tor samples have a reward value surpassing the Tor threshold. Meanwhile, the figure labeled as Non-Tor distribution showcases instances where non-Tor samples are correctly classified. Those correctly classified non-Tor samples have a reward value below the non_Tor threshold.

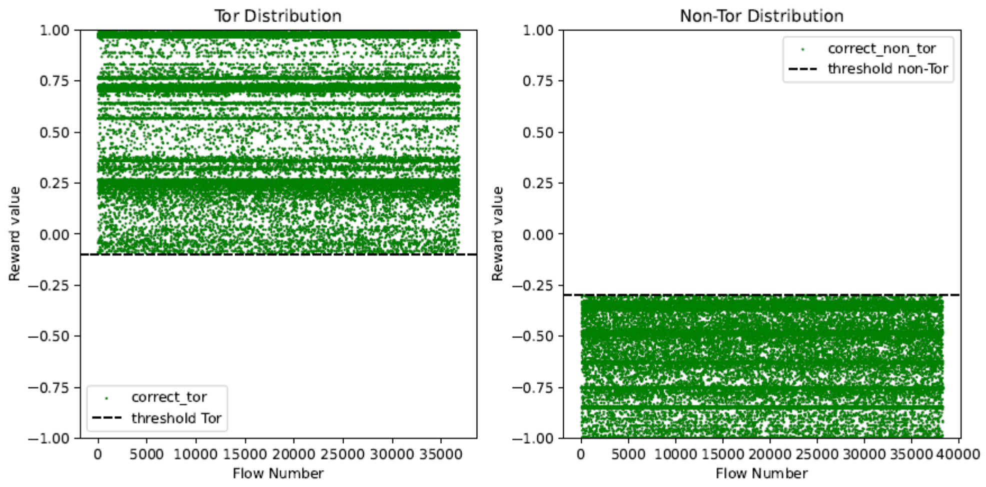

Fig. 9. Reward value distributions.

It is observed that plenty of reward values of Tor samples clustered above the -0.1 threshold. Similarly, a significant number of non-Tor samples clustered below the -0.3 threshold. Especially, reward values of non-Tor samples distributed densely below the -0.3 threshold.

The latency of the proposed system is measured. As a real-time traffic monitoring tool, the system is expected to respond promptly and make decisions within tight time constraints. To measure the latency, the time elapsed of processing 91,483 traffic flows is recorded. In this measurement, the recorded time for processing 91,483 traffic flows ranged from 1 to 1.2 seconds. This indicates that the system operates within an acceptable time frame.

4.6 Model Comparisons

The performance of the proposed system was compared with supervised baselines by accuracy. The baselines are the CNN model (DeepImage) described in [10], the stacking ensemble model developed by [12], the Random Forest Classifier in our experiment, the improved decision tree algorithm (Tor-IDS) [18], and the random forest tested on the Anon17 dataset [16]. Table 10 lists the comparison results.

Table 10
Comparison of Model Performance

Model	Dataset	Accuracy
Proposed Real-Time System	CIC-Darknet2020	0.82
DeepImage [10]	CIC-Darknet2020	0.95
Random Forest by Grid Search	CIC-Darknet2020	0.97
Ensemble Model [12]	CIC-Darknet2020	0.98
Tor-IDS [18]	Self-Collected Network Traffic	0.99
Random Forest [16]	Anon17	0.99

The comparison results indicate that all supervised models have outstanding performance in identifying Tor traffic patterns. Among those models, the improved decision tree algorithm and the random forest implementation [16] have the highest accuracy at up to 0.99, followed by the ensemble model at 0.98, the random forest classifier in our experiment at 0.97, and the DeepImage model at 0.95. Notably, the proposed model lags behind the supervised models in terms of accuracy, while it achieved an accuracy level exceeding 80%. The differences in performance can be attributed to the high efficacy of supervised models in processing static and historical data. In particular, ensemble models and random forest models have been highly effective in classification tasks based on voting mechanisms. However, it is important to underscore that the proposed model targets to operate in an unsupervised mode. This strategy addresses the challenges posed by the lack of labeled data in real-time network environments. As a trade-off, the proposed model is designed with the support of a single dataset. Furthermore, it does not rely on archived labeled data for its operation. Additionally, each of the single-layer neural networks used for emulating the reward functions has a simple structure and fast training time. With continued improvements, the accuracy achieved is considered satisfactory in this study. Ultimately, those aspects make deployments of this system to real-time situations more practical and robust.

5 SUPPLEMENTAL DESIGNS

This is a separate section that explains the issues encountered in the preliminary designs of the detection model.

5.1 Reward Function and Threshold Setting

5.1.1 Linear Regression-Based Reward Function

Due to limited information about the influence of chosen features on prediction accuracy, the raw importance scores produced by the Random Forest Classifier were used as the weights. Meanwhile, the importance scores are scaled such that they sum to 1. The reason is to emphasize the impact of feature importance on prediction accuracy. Table 11 summarizes the rank. The result is that Idle Max is ranked the most important and Idle Mean is ranked the 10th most important with a score of 0.03748. According to Table 11, the reward function is represented as

$$r = (0.05232 \times f_1 + 050376 \times f_2 + \ldots + 0.03748 \times f_{10})$$

where f_1 is Idle Max and f_{10} is Idle Mean.

Table 11
Importance Ranking by Random Forest

Feature	Feature	Importance Score
1	Idle Max	0.05232
2	Bwd Packet Length Min	0.050376
3	Fwd Seg Size Min	0.04712
4	Flow IAT Min	0.0442712
5	Flow Bytes/s	0.040979
6	Fwd Header Length	0.040552
7	Flow IAT Mean	0.040008
8	Bwd Packets/s	0.039743
9	Flow IAT Max	0.038301
10	Idle Mean	0.03748

5.1.2 Weighted Mean-Based Threshold

Built upon the linear reward function, the threshold was derived by averaging the rewards of all samples. In the experiment, we consider samples with a reward of below average as non-Tor and samples with a reward of above average as Tor. The result is that 66% of 91483 samples conform to this distribution. Since the importance scores vary slightly among trials while keeping the hyperparameters fixed of the Random Forest Classifier, it is difficult to enhance the performance by optimizing the weight values. As a result, a hyperbolic tangent transformation is added to facilitate the classification task, as shown in Equation 3.

5.2 MDP Formalization

In the initial detection model, the transition between states depends on the value of the immediate reward. However, this environment model is different from a typical transition probability, where the immediate reward is not expected to interfere with the transition probability. To address this problem, we attempted to formalize the MDP by collecting trajectories of the agent interacting with the environment. In each trajectory, the sequence of the state, action, and reward at each time step was recorded in the form of $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, ..., s_N)$. The cumulative reward along a trajectory, V, is the value of each trajectory, which is written as

$$V = \sum_{t=0}^{\infty} \gamma^{-t} \times r_t$$

By observing the values of s_i, a_i, and r_i, the agent aims to improve a policy by maximizing V, or the cumulative reward. Nevertheless, considering there is no terminal state in the network environment, the design focused on setting the thresholds properly and implementing an unsupervised detection model based on reinforcement learning.

6 FUTURE WORK

Due to constraints in developing a comprehensive detection model on a local machine and the challenges of simulating Tor networks in a cloud environment, we were unable to extract and analyze data on DNS delay time, bandwidth of Tor relays, and darknet server specifications. Given the possible misuse of the Tor network and attacker behaviors, these features are inferred to play a pivotal role in distinguishing between malicious and benign anonymous traffic. As emphasized previously, differentiating malicious from benign anonymous traffic is essential for thwarting penetrations and preserving privacy. Therefore, it is worth continuing the experiment in this track. Beyond these features, the efficacy of the proposed model can be tested on other public anonymous traffic datasets. The performance will be compared across different datasets.

What's more, within the context of internetworking, the environment model along with the dependencies among sequential flows remain unverified. In future work, we intend to examine the environment model by controlling the generation of each traffic flow. The insights will be further applied to adapt sequential decision-making frameworks to anonymous traffic detection.

7 CONCLUSIONS

Anonymous networks protect user’s identity by relaying data through a distributed network. By using tools such as the Tor network, the transmitted data are hard to read and trace. On the other hand, traffic from anonymous networks can be suspicious or malicious. Therefore, detecting anonymous traffic in real-time is crucial yet inherently difficult. Some supervised learning methods have been proposed and have demonstrated outstanding performance on static data. However, some models exhibit the shortcomings of time-consuming and low adaptability. Thus, an effective real-time detection system for anonymous traffic contributes significantly to mitigating cyber threats.

This work developed an unsupervised real-time system for detecting anonymous traffic by using a labeled dataset containing 141,530 samples. Features that are most informative about Tor and non-Tor traffic were extracted to define the tuple containing the information about the state, action, and reward. Upon deployment, the system continuously monitors TCP flows and observes relevant features. Instead of using training data, the system makes decisions by reward signals from the environment and compares each reward value with the predefined thresholds. Based on comparison results, the agent adjusts its actions and traverses to either the Tor or Non-Tor state. In the testing phase, the model was gauged in a supervised manner. The result indicates that the accuracy of the model for detecting anonymous traffic is 82%.

Literature Cited

[1] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” *Proc. 13th Conf. USENIX Secur. Symp.*, vol. 13, pp. 303-320, 2004.

[2] K. Peng, “How secure are the main real-world mix networks - case studies to explore vulnerabilities and usability,” in *Proc. 2023 ACM Asia Conf. Comput. Commun. Secur.*, 2023, pp. 539–551.

[3] R. M. Parizi, S. Homayoun, A. Yazdinejad, A. Dehghantanha, and K. R. Choo, “Integrating privacy enhancing techniques into blockchains using sidechains,” in *2019 IEEE Can. Conf. Electr. Comput. Eng.*, 2019, pp. 1–4.

[4] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of encrypted and vpn traffic using time-related features,” in *2nd Int. Conf. Inf. Syst. Secur. Privacy*, 2016, pp. 407–414.

[5] A. Taichi and G. Atsuhiro, “Graph visualization of the dark web hyperlink,” in *2020 8th Int. Symp. Comput. Netw.*, 2020, pp. 89–94.

[6] R. Biswas, E. Fidalgo, and E. Alegre, “Recognition of service domains on tor dark net using perceptual hashing and image classification techniques,” in *8th Int. Conf. Imag. Crime Detect. Prev.*, 2017, pp. 7–12.

[7] S. Kumar, H. Vranken, J. van Dijk, and T. Hamalainen, “Deep in the dark: A novel threat detection system using darknet traffic,” in *2019 IEEE Int. Conf. Big Data*, pp. 2019, 4273–4279.

[8] I. Ghafir, J. Svoboda, and V. Prenosil, “Tor-based malware and tor connection detection,” in *Int. Conf. Frontiers Commun., Netw. Appl.*, 2014, pp. 1–6.

[9] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, “Detecting malicious URLs using lexical analysis,” *Netw. Syst. Secur.*, vol. 9955, pp. 467–482, 2016.

[10] A. H. Lashkari, G. Kaur, and A. Rahali, “Didarknet: A contemporary approach to detect and characterize the darknet traffic using deep image learning,” in *10th Int. Conf. Commun. Netw. Secur.*, 2020, pp. 1–13.

[11] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, “Characterization of Tor traffic using time based features,” in *3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 253–262.

[12] H. Mohanty, A. H. Roudsari, and A. H. Lashkari, “Robust stacking ensemble model for darknet traffic classification under adversarial settings,” *Comput. Secur.*, vol. 120, 2022, Art. no. 102830.

[13] E. P. Xing. “Probabilistic graphical models.” cs.cmu.edu. Accessed: July 1, 2023. [Online.] Available: http://www.cs.cmu.edu/~epxing/Class/10708-20/

[14] K. Yu, K. Nguyen, and Y. Park, “Flexible and robust real-time intrusion detection systems to network dynamics,” *IEEE Access*, vol. 10, pp. 98959–98969, 2022.

[15] K. Shahbar and A. N. Zincir-Heywood, “Anon 17: Network traffic dataset of anonymity services,” Dalhousie University, 2017.

[16] M. Antonio, C. Domenico, A. Giuseppe, and P. Antonio, “Anonymity services tor, i2p, jondonym: Classifying in the dark,” in *2017 29th Int. Teletraffic Congr.*, vol. 1, 2017, pp. 81–89.

[17] “Tor project.” torproject.org. Accessed: Jan. 7, 2023. [Online.] Available: https://www.torproject.org/about/history/

[18] J. Lingyu, L. Yang, W. Bailing, L. Hongri, and X. Guodong, “A hierarchical classification approach for tor anonymous traffic,” in *IEEE 9th Int. Conf. Commun. Softw. Netw.*, 2017, pp. 239–243.

[19] R. Sutton and A. Barto, “Dynamic programming,” in *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: MIT Press, 2018, ch. 4, pp. 89-111.

[20] R. Sutton and A. Barto, “Temporal-difference learning,” in *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018, ch. 6, pp. 143–165.

[21] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015.

[22] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928-1937.

[23] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” in *Proc. 31st Int. Conf. Mach. Learn.*, 2017, pp. 1889-1897.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *Arxiv.*, early access. doi: https://doi.org/10.48550/arXiv.1707.06347.

[25] H. Jo and K. Kim, “Security service-aware reinforcement learning for efficient network service provisioning,” in *2022 23rd Asia-Pacific Netw. Operations Manage. Symp.*, 2022, pp. 1–4.

[26] G. Han, L. Xiao, and H. V. Poor, “Two-dimensional anti-jamming communication based on deep reinforcement learning,” in *IEEE Int. Conf. Acoust., Speech Signal Process.*, 2017, pp. 2087–2091.

[27] G. Caminero, M. Lopez-Martin, and B. Carro, “Adversarial environment reinforcement learning algorithm for intrusion detection,” *Comput. Netw.*, vol. 159, pp. 96–109, 2019.

[28] N. C. Luong *et al.*, “Applications of deep reinforcement learning in communications and networking: A survey,” *IEEE Commun. Surv. Tut.*, vol. 21, pp. 3133–3174, 2019.

[29] “scikit-learn: machine learning in Python.” scikit-learn.org. Accessed: Jan. 7, 2023. [Online.] Available: https://scikit-learn.org/stable/

[30] C. Song, W. Fan, S.-Y. Chang, and Y. Park, “Reconstructing classification to enhance machine-learning based network intrusion detection by embracing ambiguity,” *Commun. Comput. Inf. Sci.*, vol. 1383, pp. 169-187, 2021.

[31] S. Subbiah, K. S. M. Anbananthen, S. Thangaraj, S. Kannan, and D. Chelliah, “Intrusion detection technique in wireless sensor network using grid search random forest with Boruta feature selection algorithm,” *Commun. Netw.*, vol. 24, pp. 264–273, 2022.