AMERICAN UNIVERSITY OF BEIRUT

SEEING THROUGH NAT TO DETECT SHADOW IT: A MACHINE LEARNING APPROACH

by
REEM KHALIL NASSAR

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
November 2022

AMERICAN UNIVERSITY OF BEIRUT

SEEING THROUGH NAT TO DETECT SHADOW IT: A MACHINE LEARNING APPROACH

by

REEM KHALIL NASSAR

Approved by:

Prof. Ayman Kayssi, Professor — Advisor
Electrical and Computer Engineering

Prof. Imad Elhajj, Professor — Member of Committee
Electrical and Computer Engineering

Prof. Hazem Hajj, Associate Professor — Member of Committee
Electrical and Computer Engineering

Date of thesis defense: November 15, 2022

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

Student Name:	Nassar	Reem	Khalil
	Last	First	Middle

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of my thesis; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes:

- ☒ As of the date of submission
- ☐ One year from the date of submission of my thesis.
- ☐ Two years from the date of submission of my thesis.
- ☐ Three years from the date of submission of my thesis.

Reem December 21, 2022

Signature Date

ACKNOWLEDGEMENTS

Firstly, I would like to express my thanks to my patient and supportive advisor, Prof. Ayman Kayssi, for his guidance, assistance, and constructive comments throughout this research project. Furthermore, I would like to thank Prof. Imad Elhajj and Prof. Hazem Hajj who accepted to serve on my committee for extending their support.

I would like to express my thanks and appreciation to the faculty and staff of the Electrical and Computer Engineering department.

Finally, I must express my very profound gratitude to my family, friends and whoever gave me support and encouragement during my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

ABSTRACT
OF THE THESIS OF

Reem Khalil Nassar for Master of Engineering
Major: Electrical and Computer Engineering

Title: Seeing Through NAT to Detect Shadow IT: A Machine Learning Approach

Network Address Translation (NAT) is present in many routers and Customer Premise Equipment (CPEs). It is used to distribute internet access to several local hosts. Most NAT devices implement Port Address Translation (PAT), which allows mapping multiple private IP addresses to a single public IP address. The private network behind a NAT becomes hidden from the public internet and only a single outward IP address will be visible to Internet Service Providers (ISP's). With the proliferation of unauthorized wired and wireless NAT routers, internet subscribers can re-distribute an internet connection or deploy hidden devices, thus causing a problem known as shadow IT.

To this end, it is of ISP's interest to know how their services are used. This study will propose a method to detect NAT devices and identify the size of the network (number of hosts) hidden behind them. A supervised Machine Learning (ML) algorithm that uses aggregated network traffic flow features is proposed to detect NAT devices. Traffic features are aggregated within multiple window sizes to study the effect of feature aggregation on NAT detection. The host counting algorithm is processed by a machine learning approach on real network traffic features. This research demonstrates that eXtreme Gradient Boosting (XGBoost) performs best in NAT detection and hidden network size detection. Whereas the Random Forest (RF) classifier was more able to predict the exact number of hidden hosts than any other algorithm. The XGBoost NAT detection model can detect NAT devices with a 97.09% F1 score which significantly outperforms many state-of-the-art methods. The exact host counting model resulted in a 65.53% F1 score, and the result increased to 90.63% after transforming the problem into a binary one. Most previous methods focused on achieving a high detection rate on given datasets instead of focusing on the model's generalizability. However, this thesis focuses on the performance of the detection algorithms especially when the network data is subjected to intended obfuscation or even when there is an environment change. The performance of detection models dropped below 70% when testing the model in a new network environment. In this thesis we also focus on interpreting the behavior of the complex algorithm to enhance trust in the results, understand the generalizability, and explain the importance of feature aggregation in case of NAT. Two eXplainable Artificial Intelligence (XAI) methods are used to analyze the generalizability of a given feature set to different network environments or after performing obfuscation techniques. These methods are also used to study the sensitivity of the detection algorithms to the aggregated feature set extracted. Finally, this study uses transfer learning to build an optimized model that can work in case of any feature change in the network traffic data.

TABLE OF CONTENTS

ILLUSTRATIONS

Figure

TABLES

Table

ABBREVIATIONS

NAT	Network Address Translation
NATD	Network Address Translation Device
PAT	Port Address Translation
CSV	Comma-Separated Values
IoT	Internet of Things
ISP	Internet Service Provider
CPE	Customer-Premises Equipment
LAN	Local Area Network
ML	Machine Learning
SVM	Support Vector Machine
LR	Logistic Regression
MLP	Multilayer Perceptron
NB	Naïve Bayes
kNN	k-Nearest Neighbors
RF	Random Forest
XGBoost	eXtreme Gradient Boosting
XAI	Explainable Artificial Intelligence
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPID	Internet Protocol Identification Field
TTL	Time To Live
OS	Operating System
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
LIME	Local Interpretable Model-Agnostic Explanations
SHAP	SHapley Additive exPlanations
SER	Structure Expansion/Reduction
STRUT	Structure Transfer

CHAPTER 1

INTRODUCTION

The number of users connected to the Internet is increasing daily. At the start of 2022, the number of internet users reached about 4.95 billion which represents 62.5 percent of the world's total population [1]. The number of connected devices, including IoT, is estimated to reach around twenty-nine billion, and the number continues to increase [2]. To provide internet access for these devices, each must have its unique IP address. Yet, IPv4 addresses are becoming limited, and it turns out to be impossible to provide a unique IP address for each connected device. Although IPv6 is suggested to solve the address depletion problem, there is an issue in the transition to IPv6 as most of the internet traffic still carries IPv4 addresses [3].

1.1 Motivation

Network Address Translation (NAT) is a method suggested to allow multiple hosts to be connected to the internet using a single routable IP address. A NAT device connects the private network (non-routable IPs) to a public one (routable IP). It alters the source IP address of the packet received from the internal network into a registered IP address before forwarding the packet. Using an address remapping technique, a NAT gateway allows the private network to share a single public IP address [4]. Thus, internet users receive packets holding the same IP address but sent from different hosts behind a NAT. Correspondingly, these users send back packets to the same IP address. Using NAT tables, the NAT router can track all the connections and modify the destination IP address

of a packet received from the public network into the IP address of a certain host from the internal network.

Implementing NAT in most routers and customer-premise equipment (CPEs) will allow Internet Service Providers (ISPs) to grant Internet access to users even if IPv4 addresses are depleted. In addition, NAT routers can hide the identity of connected users and only the public IP address is revealed [5]. It thus provides security and anonymity by hiding the inner network topology [3]. However, this NAT property poses few problems to ISPs since it hinders their ability to know how their services are used or how large the inner network might be and manage it properly [6]. The subscriber or user can also use unauthorized NAT devices to illegally sell and redistribute an internet connection to other users. Therefore, it is of an ISP's interest to know if a single IP address might represent a large network in order to determine if it is facing a shadow IT situation [7].

Detecting NAT devices only is not enough. It is important to also identify how many devices are connected to it. Any anomalous behavior of a NAT device can be caused by one of the end hosts behind it. One of the solutions to this problem is to block the IP address that is performing malicious behavior. However, in the case of NAT, this might affect other hosts in the internal network. To manage network traffic precisely, we need to distinguish the hosts behind one NAT gateway from traffic traces with the same source IP address [8]. This study will focus on detecting NAT devices and approximating the number of hosts behind them by analyzing passively collected internet traffic flows.

1.2 Objectives

This section outlines the concrete objectives of this thesis:

1. Extracting aggregated flow features from passively collected data.

2. Evaluating different machine learning algorithms, comparing, and getting the one with the best performance on detecting NAT.
3. Comparing ML NAT detect algorithm with a state-of-art traditional method.
4. Build a cascaded classifier which will detect the number of hidden hosts from aggregated feature vectors that are predicted as data coming out from a NAT device.
5. Evaluating multiple hosts counting models and using the best model.
6. Analyzing the performance of selected algorithms on obfuscated data.
7. Analyzing the performance of selected algorithms on new data.
8. Explaining the model performance on obfuscated and new data.
9. Analyzing the model's sensitivity to the extracted network traffic features.
10. Applying transfer learning to build an optimized model.

1.3 Contributions

In this thesis, the main objective is to present a machine-learning algorithm that can identify NAT devices using time aggregated traffic flow features of locally collected data. This method does not require any interaction with end-users, respects their privacy, and is not limited to specific types of operating systems. All previous NAT detection methods depend on detecting NAT using features extracted per flow which may lead to a high false-positive rate. For example, if the algorithm is using the time to live (TTL) value as a feature to detect NAT, some NAT devices are deployed in a way that could not decrement the TTL values and thus resulting in false predictions. To decrease the false positive rate and reduce the training time, aggregated features using different window sizes are used. Aggregating the network flows per time, resulted in extracting new

features that gives high evidence for the presence of NAT and have not been used in the literature before.

To show the importance of flow aggregation per time in NAT detection, the classification results of multiple window sizes are compared. Aggregating features increases classification accuracy because it increases the chance of having more than one active user, especially when the time window size is large. Multiple machine learning algorithms are used at this stage to find the best one that can predict NATs. A comparison between the proposed technique results and those of the traditional method presented in [9] is done.

Once the NAT detection model detects the presence of a NAT, then it is time for host number identification without being restricted to a specific type of operating system (OS). In this stage, two methods are applied. Multiclass classification for detecting the exact number of hosts and binary classification for detecting the network size. Binary classification is used to avoid being limited to a specific number of hosts in detection. Similarly, the host number identification approach is processed by multiple machine learning algorithms to find the one that gives the most accurate results.

Then the focus will be on evaluating the generalizability of the models developed. Most of previous studies have focused on getting a high detection rate on a given dataset. Some have studied the model's generalizability in case the hidden hosts use different types of operating systems as in [8]. None have taken into consideration the obfuscation techniques that are applied to network traffic and thus hinders the model's ability to detect NAT devices or hidden network size. First, the model is evaluated by testing its predictions after obfuscating the training dataset. Multiple obfuscation scenarios are implemented by changing packet sizes related features, number of packets features, etc.

Then it is evaluated on new data collected in a different network environment. Then, to avoid treating the ML algorithm as a black box, understand the classification technique, analyze the model's ability to generalize, and study the usefulness of the new extracted features, eXplainable AI (XAI) is used. Finally, transfer learning is applied to build an optimized model that is able to detect NAT and approximate the number of hosts in case of any feature change in the network traffic.

The proposed approach uses aggregated features to extract added features and build a more effective and realistic detection model. It also focuses on analyzing the generalizability of proposed approaches on other data, obfuscated and new. It is the first that analyzes the effect of obfuscating features on NAT detection and number of hosts prediction. It is the first that uses XAI to explain the implemented model performance ability to generalize. Finally, it is not limited to any type of operating system or number of hosts connected.

1.4 Structure of Thesis

In the rest of this thesis, Chapter 2 summarizes the existing works in the literature. Chapter 3 discusses the dataset used in this thesis, the fundamentals of feature extraction and selection, a description of preprocessing steps, the NAT detection mechanism, the host counting mechanism, the machine learning approaches, and explainable AI methods. Chapter 4 presents the evaluation of the different techniques employed using several performance metrics with analysis, in addition to XAI results. Chapter 5 concludes and gives directions for future work.

CHAPTER 2

LITERATURE REVIEW

Despite the evidence that NAT allows for better use of the existing IPv4 address space for an organization, it is also sometimes employed by malicious internet subscribers to allow anonymous access to the Internet. NAT host identification is used to determine the number of hosts that are connected behind a NAT device. This can be valuable for network management purposes in order to get information on how many users and devices are connected to the network. There are different approaches proposed in the literature to detect NAT devices and identify the number of hosts behind the NAT. In addition, multiple approaches have been proposed to build a generalizable model and to explain any implemented "black box" model. In this chapter, some of the methods will be explained to gain a basic knowledge of how others have performed NAT detection, host detection and transfer learning, how to build a generalizable model, and how to explain and extend the machine learning black-box model. The related work can be divided into five categories (1) NAT detection, (2) NAT host profiling and number identification, (3) generalizability, (4) explainability, and (5) transfer learning.

2.1 NAT Detection

Multiple traditional methods have been suggested to identify NAT behavior. Krmicek *et. al.* [9] proposed three methods to detect NAT devices passively and reduce the false-positive rate. Their methods depend on the IP, TTL, and Identification fields, which in turn depend on previous approaches by Phaal [10] and Bellovin [6] respectively. They have also introduced new methods that depend on subnet TTL and the length of

TCP SYN packets. To reduce the false-positive rate, they aggregated the results of each implemented method. Orevi *et. al.* [11] proposed a De-NAT scheme, i.e., re-identifying the communication flowing out and into the NAT. Their method was based on IP Identification and TCP Timestamp. They limit their algorithm to specific operating systems like Windows 8, 10, and Android to use the properties of the TCP/IP stacks in these systems, such as the increment in the IP Identification field when packets are sent to the same destination. They applied their method by sending DNS requests to the same DNS resolver. What differentiates their method from previous ones is that they require only the DNS requests, not the whole traffic.

A method based on application-level presence information is implemented by Bi *et. al.* [12]. Their method takes advantage of application layer fingerprinting where NAT gateways do not modify application layer information. Their algorithm is designed for NAT-aware routers that are usually used by ISPs to detect NAT gateways. They can detect NAT gateways based on the presence of fingerprints of instant messaging applications, the IP address of the application server, the registered TCP/UDP port numbers of the servers, and the specific format of certain instant messaging packets.

Another passive method that is based on NetFlow data only was suggested by Yan *et. al.* [13]. This method relies on the Out-In Activity Degree for given network behavior. It shows enormous success in a large-scale network, where it reported an accuracy of 92%. Lutu *et. al.* [14] proposed NAT Revelio to detect large-scale NAT as carrier-grade NAT and large-scale NAT without any previous knowledge of the testing environment. Their method is applied by performing six active tests against different elements, including one or more servers deployed on the public Internet.

Present NAT detection work is mostly based on machine learning approaches. Komarek *et. al.* [15] implemented an algorithm that detects NAT behaviors by using the IP-based features presented in HTTP access logs. These features include the number of (1) unique communicated IP addresses, (2) persistent connections, (3) unique operating systems and versions, (4) unique user agents, (5) unique browsers and versions, (6) downloaded bytes, (7) uploaded bytes, and (8) sent HTTP requests. They employ linear machine learning models, support vector machine (SVM), and logistic regression (LR), to find hidden patterns in the statistical data and thereby detect the presence of NAT. To have sufficient data to train their classifiers, they have generated artificial NAT traffic data by merging HTTP logs of multiple hosts. Their results show 94.75% accuracy with cross-validation. Although they have reached satisfactory results, as any other ML algorithm their classifier needs a lot of known samples to be well trained.

Another machine learning algorithm that is based on statistical features derived from NetFlow is proposed by Abt *et. al.* [7] who used the same statistical features in [9]. Their approach was capable to work in real-time and achieves an accuracy of 89.35% on passively collected data. Yan *et. al.* [16] suggested a novel method that can detect large-scale NAT using a semi-supervised deep neural network. They aimed to identify NAT for Internet of Things (IoT) devices. They applied their method to a small dataset with features extracted from network, transport, and application layers. After implementing and testing their method, they applied it to a real-world dataset and achieved up to 92% of precision and recall.

Khatouni *et. al.* [17] proposed a machine learning approach to detect NAT using statistical features in passively collected flows. They also proposed an approach that detects NAT using aggregated application layer information as different browser versions

and several types of operating systems in each time window. They have applied multiple machine learning algorithms on traffic features extracted by multiple network traffic analyzers (NTA). Their results show that the decision tree (DT) classifier applied on features extracted by Tranalyzer NTA is the best. But their method suffers from the limitation that if all hosts use the same operating system, NAT cannot be detected. Also, if a device has two different versions of the same operating system this might result in two different browser versions and detect them as two different browsers and falsely identify them as NAT.

2.2 NAT Host Profiling and Number Identification

Multiple methods have been proposed in the literature to detect the number of hidden hosts. Previous studies have either proposed traditional methods or machine learning algorithms to count or identify hidden hosts. Bellovin [6] implemented one of the first traditional methods to count the hidden hosts behind a NAT device. His technique was passive and based on the IP header's identification field (IPID). This method relies on the observation that the host in many operating systems increases the IPID field by one when it sends a new packet. This method can work only with specific operating systems and is limited in the case of randomized IPID generator as in FreeBSD OS.

Another passive method that identifies hosts behind a NAT using IPID, SYN flag, TTL, and timestamp is proposed by Park *et. al.* [5]. This method was able to count the number of hidden devices and identify their OSs effectively. Yet, this method is limited to a specific number of OSs. Their method shows the highest accuracy when the hidden hosts are Linux-based and the lowest accuracy when OS is Windows. They reported a total average accuracy in their experiments of 84.44%.

Few studies proposed machine learning algorithms to count and identify hosts. A machine learning algorithm was proposed by Lee *et. al.* [18]. They proposed an actively supervised learning method based on port patterns. Their method can work remotely, and it achieved an F1 score of 90%. After identifying NAT devices, the behavior and status of the connected hidden hosts behind each NAT Device (NATD) can be investigated by monitoring packets with IP addresses for NATDs, not capturing all packets as in conventional methods. Rui *et. al.* [19] proposed an algorithm based on analyzing traffic with a Directed Acyclic Graph Support Vector Machine (DAGSVM) to detect hosts hidden behind a NAT device. Their method was composed of four steps. First, they prepared the traffic flow features for the SVM classifier. Then they trained the DAGSVM classifier to data containing *n* hosts. Third, the used features are linear thus they collected the traffic of one host and then get traffic models of *n* - 1 hosts. This process was made to predict data with more than *n* hosts. Finally, the model classifier can calculate the number of hosts. Their algorithm was successful in predicting the number of hosts greater than that in training data. However, the model started to fail with increasing number of hosts.

Shukla *et. al.* [20] proposed a machine learning algorithm that can identify and count the total number of hosts masqueraded by a single IP address. They aimed to build a model that can identify if a hidden malicious behavior is out from a single or multiple hosts without being dependent on OS, IP addresses, and port numbers. Eight multi-class machine learning classifiers are trained on non-NATted traffic from 1116 hosts. This experiment aimed to find the best ML algorithm that can identify patterns in network traffic. RF classifier and decision tree outperform all other used algorithms. The RF classifier yields the highest results. So, it was used as a base model for their algorithm.

Each host is identified by its unique source IP and then marked as a unique class. Then they trained the model on data outward from multiple ordinary hosts and tested on NATted traffic data, and NATted and non-NATted malicious traffic. The model performance drops when testing it on a new dataset, but it is still able to count hosts. The algorithm reached an average testing performance of 91% when all datasets are used. Besides their algorithm outperforms the state of art signature-based approach and obtained a score of 66.66% which is higher than other benchmark studies.

An algorithm that incorporates both supervised and unsupervised learning approaches to count and cluster network traffic was introduced by Mateless *et. al.* [21]. The network traffic features used are based on characteristics of operating systems, NAT behavior, and users' habit. The algorithm is not only implemented on traffic data out from physical devices as Windows, Linux-based, iOS, and Android, but also on traffic data outward from containers, virtual machines, and load balancers. Counting the network entities behind a NAT is performed using a dedicated algorithm. Then the traffic features and the number of network entities counted are fed into a multiple clustering algorithms. They have applied the clustering algorithms to each type of OS separately and found that clustering Linux-based hosts are easier than Windows hosts. Their algorithm achieved 87%, 81%, and 100% F1 measure for Linux, Windows, and containers on virtual machines respectively. Yet their model is incompatible with traffic modification, and when network traffic is a combination of multiple OSs.

2.3 Generalizability

It is significant to apply NAT detection methods in the real world, thus, to be able to apply them the model generalizability must be achieved. Gokcen *et. al.* [22] suggested

multiple machine learning approaches that can identify NAT-like behaviors by exploring specific patterns in the network traffic. They use NetMate2 to generate flows and exclude port numbers, IP identification, and payload information. To achieve a well-generalized classifier, they tested their algorithm on two different datasets. Yet the generalizability of their approach is still limited since they have built one model that is trained and tested twice. To illustrate, the implemented model is trained and tested on the first dataset, and then to prove its generalizability it is trained and tested again on the second dataset. They have reached a NAT class detection rate of 98.7% for the first data set and 98% for the second dataset using C4.5 algorithms. However, the detection rate dropped to 15% for the first dataset and 34% for the second one using a naïve approach.

A NAT detection and host identification generalized approach was proposed in Zhang's thesis [8]. She examined the generalization of her proposed approach by comparing the classifier performance when trained on a specific dataset and tested with data taken from another one. First, she implemented three classifiers that are trained and tested on the same dataset (80% of data taken for training and 20% for testing). After reaching the best model, to study generalization, she performed experiments on the proposed model by training it from data in a specific dataset and testing it on data from another dataset. The implemented model achieves an accuracy of 100% when trained and tested on the same dataset, however other experiments reported lower accuracy (when tested on another dataset). In generalizability tests, the accuracy dropped to 25%, 57%, 84%, and 86%, and surprisingly stayed at 100% in one case. The 100% was reported when the model was trained on a dataset with only Kali hosts behind the NAT and tested when only Windows hosts are behind the NAT. Then a host identification technique using TCP Timestamp is applied to the NAT detection dataset. However, they tested their

model on datasets that include only Kali and Windows operating systems and each dataset is traffic data collected over one day.

2.4 Explainability

Although explainable AI (XAI) is used to explain a black-box model, it has not been used before in NAT detection or host counting to explain how the model can take a decision. Arrieta *et. al.* [23] show that multiple methods can be used to explain the decision of black-box models. These methods can be divided into model agnostic that can be applied to all machine learning algorithms or model-specific implemented to explain the decision of a specific approach. In addition, they showed that these methods can work in several types of data such as tabular, graphs, images, etc., and can be divided into local XAI which can explain the decision of one entry, or global which explains the decision of the whole model.

XAI methods have been previously used in security for such reasons as enhancing trust management in intrusion detection systems and enabling generalizability in each network environment. First, Mahbooba *et. al.* [24] have addressed the problem that AI cybersecurity models are becoming more complex. They have focused on enhancing trust in intrusion detection so that they can be understood by human experts easily. They have used the model agnostic XAI, by explaining these models by simplification. To do that they have performed feature engineering and built a decision tree (DT) model. Then they interpreted the feature importance based on DT entropy, and the rules they get from DT intrusion detection classification. Comparing their results to the state of art algorithms in the literature (SVM and logistic regression) show that DT has higher performance. To enable generalizability in securing IoT networks, Serhan *et. al.* [25] proposed an

explainable machine learning-based network detection system. They compared two feature sets (NetFlow and CICFlowMeter) in three datasets in different network environments. They showed that the NetFlow feature has better performance on ML model detection accuracy. To interpret the classification of ML models they have used Shapley Additive exPlanations (SHAP), which is an XAI method. They have compared the mean Shapely values, average values for each feature among all test samples, of different features in the three datasets to know the influence of each feature on the final model decision. Visualizing Shapely values for features identifies the features used by the model to make predictions. The importance of a key feature is indicated by how large its Shapely values are. Through SHAP they can detect whether the classifier is using the idle-based features in the attack detection stage. If the model decision was based on idle-based features, they considered that the model withholds key security events to aid the detection performance.

2.5 Transfer Learning

Transfer learning was proposed to assist in building more accurate models in a certain domain by using knowledge from the source domain. Despite the fact that transfer learning has enormous applications in deep learning like natural language processing (NLP) [26], [27], some studies have utilized it for network security-related tasks. None of the previous studies used transfer learning to improve the detection of NAT devices and hidden network size. Zhao *et. al.* [28] applied transfer learning to detect unknown network attacks. They have proposed an approach called clustering-enhanced transfer learning to automatically find the relation between a new attack and a known attack. They have used multiple classification models such as DT and RF and evaluated the novel

transfer learning approaches by assessing the model performance on a dataset that contains different attack types or subtypes from training data. Their algorithm was efficient and validated the usefulness of transfer learning in discovering previously unseen attacks. Zhao *et. al.* [29] implemented another transfer learning method. In this study, they aim to use transfer learning to predict new attacks that are not present in the training data. Their algorithm was based on optimizing the representation of the model to be invariant to the modification of attack behaviors that are presented in the training set. This technique can be used with any common-based classifier. The algorithm was successful in identifying new attacks, but it depends on manual pre-settings of hyper-parameters.

2.6 Summary

The approaches presented in this section are either signature-based which look for specific fields in IP, TCP, and HTTP header fields, or behavior-oriented that examine how the traffic from a NAT behaves as in Komarek *et. al.* [15]. The signature-based uses the special fields to gain information about the hosts behind a NAT as their number or type of operating system. These fields can also be used as an indicator of the presence of NAT. The behavior-oriented examines the network traffic and deduces patterns that indicate the existence of NAT devices. The most commonly used features are the number of packets sent and received, number of DNS requests, and number of bytes sent and received. In both methods, the accuracy of the implemented algorithm increases when there exist more active nodes hidden behind a NAT. Unfortunately, some of these methods were limited to the case where the traffic is unencrypted, and machine learning algorithms has less limitations in NAT detection than in counting the number of hosts.

They have not taken into consideration that aggregating network traffic will result in more features that have higher relation with NAT presence. Almost all the previous academic research was not deployed in real network environments. Instead of focusing on the model's generalizability of these models, studies rather focused more on the performance of the detection algorithms on a given dataset. The authors in [8] and [23] depended only on testing or even training their model on a new dataset. They have not taken into consideration that if the model generalizes well on a new dataset this does not mean that it will generalize in all other datasets and can be practically implemented. They also have not taken into consideration that intentional obfuscation of some key features in the dataset may also affect the model performance and generalizability. In addition, they treated ML-based models as a black box, so they do not often take into consideration the importance of interpreting the behavior of the complex algorithm to enhance trust in the results and understand the generalizability. EXplainable Artificial Intelligence is suggested as a solution to interpret decisions taken by ML techniques. Through understanding what features are contributing to the model decision, explainable AI can help in maintaining and troubleshooting the practical implementation of these models. Finally, although transfer learning has multiple successful applications and is considered a promising area of machine learning, none have applied it to optimize model performance in a new network environment.

CHAPTER 3

METHODOLOGY

In this chapter, first, the datasets used to train and test the implemented models, and the network configuration are explained. Then the implementation of the NAT detection approach along with the generalizability test, explainability, and transfer learning are discussed. This is followed by presenting the methodology and details of the host counting algorithm.

3.1 Datasets

3.1.1 Wireless Setup Dataset

To identify NAT behavior, a publicly available data set [30] is used. The dataset, "NAT Network Traffic Dataset," is 20.4GB and composed of 294 capture files from 294 tests. It consists of 112306 unique flows, thus considered large and permits us to accomplish an extensive assessment of several NAT detection approaches. Because we are working with aggregated flow features, these unique flows are aggregated within specific time window, yielding to 3497 aggregated samples at time equal 1 minute. The dataset was collected over two weeks in June 2020, with three sessions per day. Each session, morning, midday, and evening, consisted of seven different tests done by varying the devices connected to a NAT router and the application opened by devices behind NAT. The data set is labeled by the time and day of performing the test in addition to the number of devices connected along with the IP addresses and process name. Figure 1 presents the experimental configuration for a local area network (LAN) with a NAT device. The tests are divided into two configurations: (1) Network with NAT, where the

NAT router is connected to a home router and then it is distributing an Internet connection to multiple devices with different operating systems to generate more realistic traffic data, and (2) Network without NAT. A full explanation of the data set is presented in [30]. For the rest of the thesis, we will refer to this data as “Wireless NAT Dataset”.

To study the generalizability of the model when network data is subjected to intended obfuscation, multiple pattern-based features, such as features related to packet timing and sizes, are changed. Packet size-related features, time-related features, and number of packets related features are changed because they are highly related to the environment. To do this different obfuscation techniques like randomization, adding dummy bytes, padding, or even uniform features obfuscation are applied. After obfuscating all these features, multiple datasets are obtained.

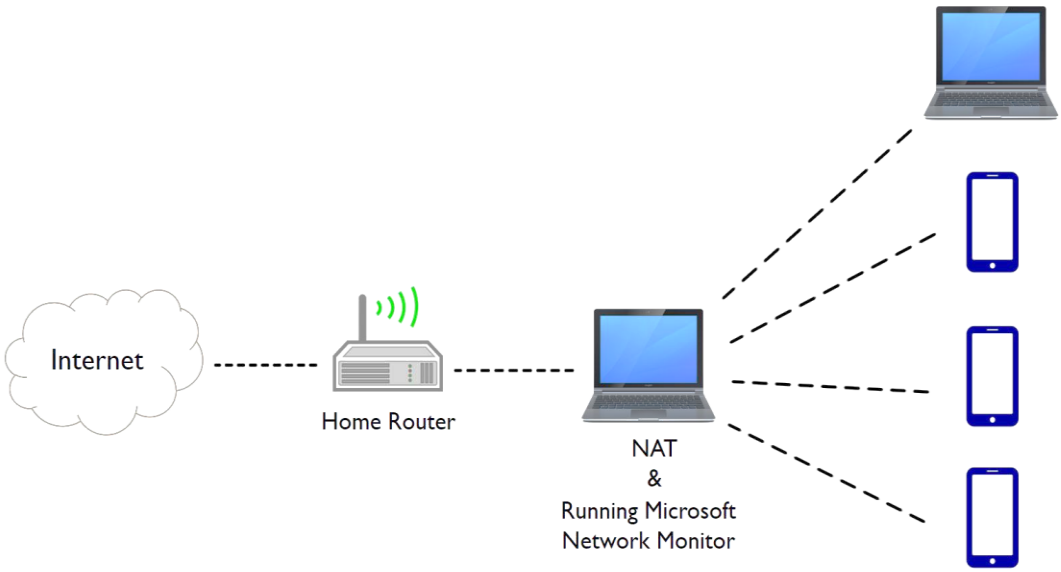


Figure 1. Passive Wireless Measurement Network Setup

3.1.2 Wired Connection Dataset

Another wired local area network was built to test the proposed detection and host number identification approach’s ability to generalize. This setup consists of a Cisco 1800 router, a Cisco catalyst 2960 switch, two Windows devices, and two Linux devices, in

addition to a kali Linux desktop that acts as a NAT device. Figure 2 presents the complete setup to capture network traffic data. The dataset is 11.7GB and 79664 unique flows are used in testing. These unique flows are aggregated, and 980 samples are used in testing. Data was collected during three days in April 2022, with different combinations of hosts running and tasks opened on each. This dataset is collected within a different network environment, different setup connections, different devices, operating systems, and different applications opened. The OS used in this dataset are Windows desktops (windows 7 and windows 11), and Linux desktops (CentOS and Ubuntu). Varity of applications are opened as gaming, shopping, social media, mail, video streaming, and google search. This difference is made to examine the generalization of the proposed method and describe the impact of different environments on NAT device detection and host number identification. For the rest of the thesis, we will refer to this data as “wired NAT Dataset”.

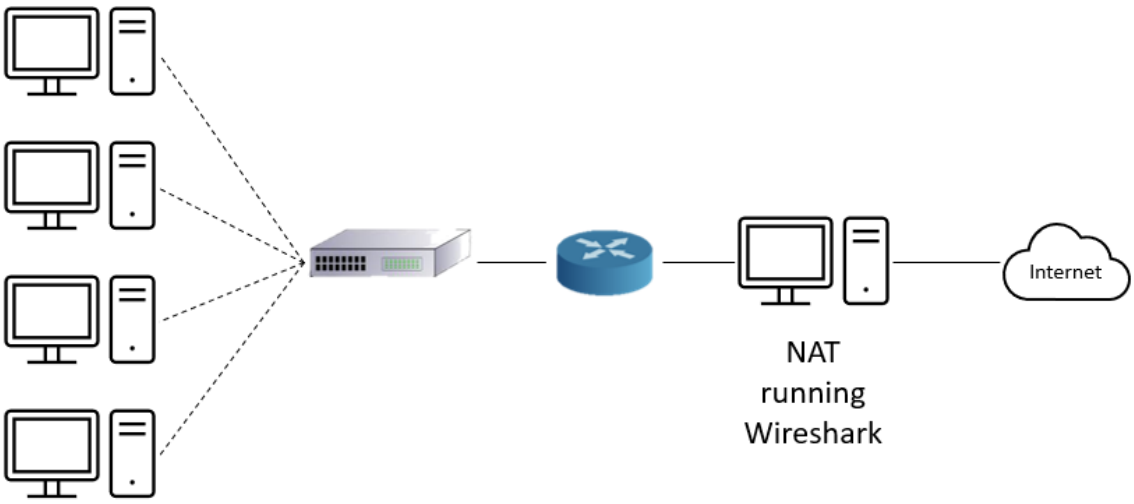


Figure 2. Passive Wired Measurement Network Setup

3.2 Feature Extraction and Preprocessing

The captured traffic traces should be preprocessed to extract features and build the detection classifiers. To this end, the Python library “Scapy” was used to generate the

flows from the capture files and compute the statistical features. The flow is identified by the tuple source IP and port, destination IP and port, and protocol. Statistical features are extracted from packet traces and then flows are formed based on the tuple. A comma-separated values (CSV) file is generated to store the flows, with the statistical features listed in Table 1. Then these features were aggregated in different time window sizes based on the flow start duration to extract the features presented in Table 2. Table 2 contains more features in which some of them have not used before as the interarrival timing between flows and the number of flows sent in a specific window size. In case of NAT detection, more flows are sent and the timing between them is less because multiple devices are active. In addition to the packet, size, and other timing related features extracted per specific time window size instead of per flow. When all aggregate features are extracted, the data is transformed into CSV files. Since supervised machine learning is used, the features that represent data coming from NAT are labeled as NAT, and the other features are labeled as not NAT. This procedure is followed for both datasets introduced in section 3.1.

TABLE 1: Features Extracted per Flow

Features	
Protocol	Smallest Packet Size
TTL	Total Packets
Source IP	Total Packets Sent
Source Port	Total Bytes Sent
Destination IP	Total Packets Received
Destination Port	Total Bytes Received
Total Packet Size	Flow Duration
Largest Packet Size	Interarrival Time

TABLE 2: Aggregated Features

Number	Feature Name
1	Number of Unique Source IP during the time window
2	Number of Unique Source Port during the time window
3	Number of Unique Destination IPs during the time window
4	Number of Unique Destination Port during the time window
5	Number of Unique TTL Value during the time window
6	Total Number Packets during the time window
7	Average Number of Packets during the time window
8	Total Bytes during the time window
9	Average Bytes during the time window
10	Total Number of Packets in the Forward direction during the time window
11	Average Number of Packets in the Forward direction during the time window
12	Total Bytes in the Forward direction during the time window
13	Average Bytes in the Forward direction during the time window
14	Total Number of Packets in the Backward direction during the time window
15	Average Number of Packets in the Backward direction during the time window
16	Total Bytes in the Backward direction during the time window
17	Average Bytes in the Backward direction during the time window
18	Total Number of TCP Packets during the time window
19	Average Number of TCP Packets during the time window
20	Total Number of UDP Packets during the time window
21	Average Number of UDP Packets during the time window
22	Number of DNS Requests during the time window
23	Size of Largest packet during the time window
24	Size of Smallest during the time window
25	Minimum Flow Duration during the time window
26	Maximum Flow Duration during the time window
27	Mean Flow Duration during the time window
28	Standard Deviation of the Flow Duration during the time window
29	Minimum amount of time between two packets during the time window
30	Maximum amount of time between two packets during the time window
31	Number of Flows during the time window
32	Maximum amount of time between two flows during the time window
33	Minimum amount of time between two flows during the time window
34	Mean amount of time between two flows during the time window
35	Standard Deviation of the amount of time between two flows during the time window

3.3 NAT Detection

This section presents the detailed methodology implemented to detect NAT. In this study, the aim is to evaluate different machine-learning-based approaches to different traffic flow features aggregated within multiple window sizes. This process aims to differentiate between a NAT device and an end host and study the effect of increasing time window size for aggregation on NAT detection.

The machine-learning-based approaches used in detection are Random Forest (RF), Multilayer Perceptron (MLP), Naïve Bayes, k-Nearest Neighbors (kNN), eXtreme Gradient Boosting (XGBoost), and SVM. The NAT identification is considered as a binary classification problem with two labels, NAT or other (not NAT), thus the machine learning classifier has two classes. The RF classifier is chosen because it is easy for human experts to understand how this algorithm takes the decision. RF builds multiple decision trees using the if-then-else format and then merges the decisions of those trees to get more stable and accurate predictions. As for Naïve Bayes, it requires less training data and can predict the class of test data easily. kNN is used because it is considered one of the simplest approaches in machine learning algorithms and it is mostly used for classification. SVM shows its success in classification problems especially when classes are not linearly separable. MLP has the advantage of solving extremely complex problems by connecting many perceptrons. XGBoost is used due to its high accuracy, efficiency, and feasibility. It is a fast algorithm able to do parallel computation on a single machine using both tree learning algorithms and linear solver model.

To train the ML algorithms the network traffic numerical features are preprocessed and represented by a feature vector. The vectors that represent the traffic consist of two kinds of sources: (1) an ordinary host, and (2) a NAT device.

For each algorithm, we build several models by varying the aggregation window size. To get more precise results we perform n-fold time-series cross-validation, where the dataset is divided into n subsets, where n is six. We perform nested time-series cross-validation to separate training and testing data by time instead of randomly to get more precise results. Since we have a dataset collected over two weeks and separated by days it was easier to split the dataset. The nested time series cross validation is used because it provides an almost unbiased estimate of true error. This method is applied through starting with a small subset of data for training purpose (6 days for training) and check the model accuracy for the later data points. Then these data points are included as a part of the next training dataset in the next fold and another following subset is used for testing. In this way we will guarantee that the model is predicting the labels on an unseen data which is collected in a different day with different devices and applications opened. This way in cross validation will enhance the model's generalizability. We evaluate our model based on the average performance of all predicted classes in cross-validation.

3.4 Host Number Identification

After predicting the presence of NAT devices, the aim now is to identify the number of hosts hidden behind them. This algorithm has great implications to understand the occurrence of hosts masked behind NAT devices for Internet service providers. To get the best model that can detect the approximate number of hosts hidden, multiple machine learning algorithms like RF, SVM, Logistic Regression (LR), kNN, XGBoost, and MLP are trained. The host counting problem is treated as multiclass classification and as a binary classification problem. First, the models are trained on data to detect the exact number of NAT devices (1,2,3, or 4). Then the problem is transformed into a binary

classification problem. The aim now is to detect the hidden network size where when there are a few devices hidden behind a NAT the label is changed to “Small NAT” else it is changed to “Large NAT”. This transformation will give evidence of the approximate number of devices, rather than the exact number of devices and it will not be limited to only four devices. Similar to NAT detection, n-fold nested time series cross-validation is used.

3.5 Machine Learning Algorithms

The detection and host number identification algorithms need to be efficient and classify classes correctly. The main goal behind this study is to detect shadow IT, thus, to detect it effectively multiple machine learning algorithms as mentioned in sections 3.3 and 3.4 are applied. The algorithm that performs the best on the NAT traffic data is chosen. The models are implemented in a python environment which allows access to powerful artificial intelligence (AI) and machine learning libraries and frameworks. The libraries and frameworks provide easy access to classification, regression, clustering, and analysis. Using python libraries, we can implement a robust machine learning algorithm. All ML algorithms are implemented using scikit-learn library except for the XGBoost, the xgboost library is used.

3.5.1 RF

The RF classifier is an ensemble-based learning method that combines multiple decision tree models during its training process to yield an optimal predictive model [31]. It is widely used in classification problems due to its simplicity in implementation and fast operation [32]. The main reason that it is used is that it does not undergo overfitting

as it takes the average of all predictions of the combined decision trees. During the training phase, RF applies the so-called bagging technique. To illustrate, RF selects random samples from the training set and fits the trees with these samples. It repeats the above procedure by replacing the chosen samples with others [33].

3.5.2 SVM

In classification, the main objective is to reach a model that has maximum performance on both training and test data. However, most of the traditional methods previously used in classification suffer from overfitting. The main idea of SVM is to build a model that separates classes in the training set by finding a hyperplane that can maximize the margin between them to avoid overfitting [34].

3.5.3 MLP

MLP is a well-known neural network that consists of three layers. The first layer which is the input layer receives the input features to be processed. The second layer which is the hidden layer is the computational engine of the algorithm. The final layer which is the output layer is where prediction and classification take place. MLP reduces the error through forward and backward propagation [35].

3.5.4 Naïve Bayes

Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that each one of the features used in training has an independent and equal contribution to the predicted class. It is simple to implement and computationally fast. It is considered a basic classification approach [36].

3.5.5 kNN

The basic principle behind kNN is that it separates instances in a dataset based on equivalent properties shared between them. Thus, the additional information of any instance can be taken from points near it. In this algorithm k is an adjustable parameter and it is defined as the number of neighbors [37].

3.5.6 XGBoost

Extreme Gradient Boosting is a supervised machine learning algorithm that tries to accurately predict the label of a vector by merging the estimates of multiple models [38]. It is used for both classification and regression problems. The XGBoost algorithm is composed of multiple trees [39]. The residual trees are built by computing the similarity score between leaves and the forecasting nodes to determine which variables are utilized as the roots and the nodes [40].

3.5.7 LR

Logistic regression is a machine learning algorithm that predicts a discrete outcome by evaluating the probability of an event occurring. It uses the logistic sigmoid function to get the probability value and map it to the predicted class. It is an easy algorithm to implement but has a high probability of underfitting to occur [41].

3.6 Performance Metrics

Instead of evaluating the approaches using the model accuracy, this study uses precision, recall, and F1-score as the evaluation metrics. It does not rely on model accuracy because it is a measure of the positive rate, and we are dealing with an

unbalanced dataset thus predicting the majority class and misclassifying the other might yield high accuracy. For multiclass classification, a confusion matrix is used to estimate the efficiency of the model in classifying each class.

3.7 Generalizability Test

This generalizability study aims to test the model's ability in detecting NAT devices and host numbers when the dataset is subjected to intended obfuscation or even when the data is collected in a new network environment. For this reason, both detection and counting models first are tested on obfuscated datasets. The datasets are generated using different scenarios after changing the pattern of the traffic like the distribution of the number of packets, packet sizes, and timing-related features as discussed in section 3.1.1. Besides the model is tested on the newly collected wired NAT dataset. This data is also subjected to intended obfuscation. In the generalizability test, we train the model on wireless NAT dataset without obfuscation and test it on other data samples.

There are multiple obfuscation techniques that could be applied on traffic data which will lead to network traffic feature change. Obfuscation techniques could lead to a single feature change or multiple feature change. The obfuscation is done uniformly, by randomizing features, adding dummy bytes, or by padding bytes. First, a single traffic flow characteristic is obfuscated. For example, when choosing packet sizes, all features related to packet sizes presented in Table 2 are obfuscated. Then because the obfuscation scenario cannot be predicted and to build a model that can generalize even when obfuscation is used, multiple features are obfuscated together. To illustrate, the obfuscated traffic data has all features related to timing, packet sizes, and number of

packets obfuscated together. Thus, forming multiple datasets with features obfuscates but each time using specific type of obfuscation, as randomizing features for example.

Then both NAT detection and host number identification models are trained after removing all features that can be affected by obfuscation. This step has been done to build a model with a high detection rate and less dependence on features that are highly related to the network environment or to mutable features that can be affected by obfuscation techniques.

3.8 Explainability

ML-based NAT detection and host number identification methods are a solution that simplifies building a detection algorithm without thinking about specific patterns in features that characterize NAT traffic data. They are both sophisticated and black-box models that achieve a high detection rate on specific datasets. Besides, most ML models are complex and hard to understand. XAI is used to enable the understanding ability of a model to generalize. Applying several methods in XAI will allow us to explain the model's local and global decisions. Understanding the rules taken by a model to make decisions and the feature contribution will make evidence of model generalizability. By using the global and local explanations we can see the importance of the extracted feature and their contribution and relation with the NAT presence. The explainability models are implemented using the open-source library by Microsoft called InterpretML [42]. For the XAI approaches, Shapely Additive exPlanations (SHAP) is used to see how the model is choosing features in order to perform detection. SHAP will help in understanding the feature importance taken by the model on each dataset, and how each feature is affecting the model's decision. SHAP can be used to explain both local and global decisions. The

Local Interpretable Model-Agnostic Explanations (LIME) method is used in order to explain the local decision of the model. LIME shows how each key feature value affects the decision for the data sample.

3.9 Transfer Learning

Different network environments have distinct characteristics. It is hard to find two network environments with similar traffic performance. For this reason, transfer learning is used to optimize model performance and allow rapid progress when testing in a new environment. Assuming that the new network environment data is a variation from the source data, a decision forest model M learned within a "source" domain can be refined using a training set sampled from a "target" domain. The model is optimized by trying to locally expand or reduce the tree around individual nodes for each tree structure and to modify the parameters associated with decision nodes. The transfer learning for RF implemented by Segev et. al [43] is used. This RF transfer technique uses two algorithms to build a learned model that can classify test data. First the RF model is implemented and trained on the source data. Then a small data portion from the target domain is taken to modify the RF classifier. First structure expansion/reduction (SER) algorithm is used. This algorithm searches greedily for locally optimal modifications of each tree structure by trying to locally expand or reduce the tree around individual nodes. The SER algorithm first will fit the target datapoints to the decision trees implemented by RF model. Then it will find the leaf where each target data point ends up. A new tree is implemented, classification error is computed, if the error is small the new tree is merged with the source model tree. This algorithm is applied on each tree in the RF model. Later the structure transfer (STRUT) algorithm is employed to adapts each tree trained on source domain to

target samples by setting new thresholds to the tree nodes. Another transfer learning algorithm is implemented for the XGBoost model. This technique will change the trees and their similarity score in the implemented model based on the target data. This will build an XGBoost model with same hyperparameters, same number of trees, but the trees in the model tweaked in order to enhance its performance on a new unseen data.

To apply transfer learning a small sample of data called “target” is taken from the data collected in a wired connection setup. This data is taken in to modify the model trained on the source data and then test the transferred model on the data collected from the new network environment.

3.10 Summary

In summary, the first stage of this research was downloading the dataset. The used dataset was then preprocessed, and features were extracted. The extracted features were aggregated within multiple window sizes. The extracted features were also used in the supervised NAT detection classification and host counting tasks. Then the implemented model was tested using the performance metrics. The dataset was finally obfuscated, and new data was collected to test the model’s generalizability. XAI was used to help in explaining and analyzing the generalizability through explaining local and global decisions taken by the model and the feature importance. Finally transfer learning is applied to enhance the model performance to new environment. The above procedures are used in the experiments found in the next chapter.

CHAPTER 4

EXPERIMENTAL RESULTS AND EVALUATION

This chapter presents the experiments that were conducted along with their results and a discussion of those results. Section 4.1 presents the feature selection for training the machine learning algorithms. Section 4.2 explains the hyperparameter tuning for each machine learning algorithm. Section 4.3 discusses the NAT detection algorithm for model selection, comparison, generalizability test, model decision explanation, and transfer learning. Section 4.4 introduces the NAT host counting algorithm for model selection in multiclass classification as well as in binary class classification, generalizability test, model decision explanation, and transfer learning. Section 4.5 summarizes the chapter.

4.1 Feature Selection

The features presented in Table 2 are used to train the machine learning algorithms and to predict classes. To guarantee that the machine learning model is performing well, a good feature selection must be done for each algorithm. The feature importance is computed through information gain, where the decrease in entropy from the transformation of the dataset is evaluated. The information gain of each variable concerning the target variable is evaluated to compute the importance of each feature and select the key features to train the machine learning models on.

4.1.1 Feature Selection for NAT Detection Model

Figure 3 illustrates the importance of the features presented in Table 2 for the NAT detection algorithm. It shows the feature name on the y-axis and the importance

parameter on the x-axis. The most important features in NAT detection are the number of DNS requests, min flow duration, min interarrival between packets, number of unique source IPs, number of unique TTL values, and mean interarrival between flows. To illustrate, the number of unique source IPs will be one if the traffic is coming out from NAT device whereas it will be equal to the number of ordinary hosts if the traffic is not coming out from NAT. The features whose importance is high, as presented in Figure 3 can be highly related to the presence of NAT. To illustrate, when there are multiple hosts behind a NAT, there are more flows recorded in a specific time window each with a different duration, interarrival between packets, and number of DNS requests. Thus, when aggregating these features and taking the total or the minimum of a specific feature there is a high chance that the machine learning algorithm will find patterns in these features that are highly related to the target. Similarly, for the interarrival between flows, more hosts mean more flows are sent from different users which will minimize the interarrival between flows in a specific time window. The number of packets have a high impact on the model's decision, and it is known that when NAT devices hide multiple active users, more packets will be sent and received. Figure 3 shows that unique destination IPs, total bytes, total bytes sent, the average number of TCP packets, maximum flow duration, and maximum interarrival time between packets are not of much importance. The maximum flow duration is not considered important in the case of NAT because while collecting the dataset in [30] there is a maximum time limit of 5 minutes where the applications used are still opened in case of the presence or absence of NAT.

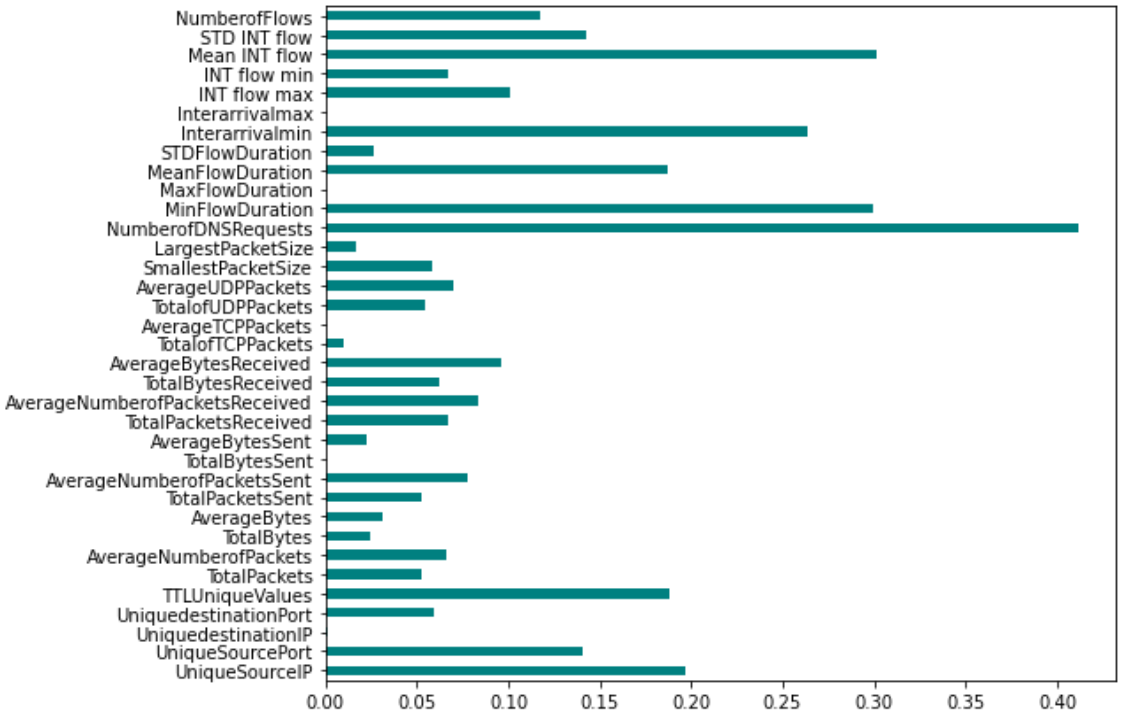


Figure 3. Feature Importance for NAT Detection Algorithm

4.1.2 Feature Selection for Multiclass Number of Host Detection Model

For the multiclass problem, it is important to guarantee that all features used by the model are related to the task and have high importance. Figure 4 shows the importance of features in Table 2 in the host counting model. The figure shows that Interarrival min, Number of DNS Requests, Largest Packet Size, Total TCP Packets, and Average TCP Packets do not affect the model decision because they have low importance. Other features in the dataset are considered important and will help the model in making correct decisions. For example, Number of Flows is considered an important feature, and this is logical because when more hosts are behind a NAT, more flows will result. Similarly for the number of packets and bytes, when there are more hosts more packets will be sent which will lead to a higher total or average number of packets.

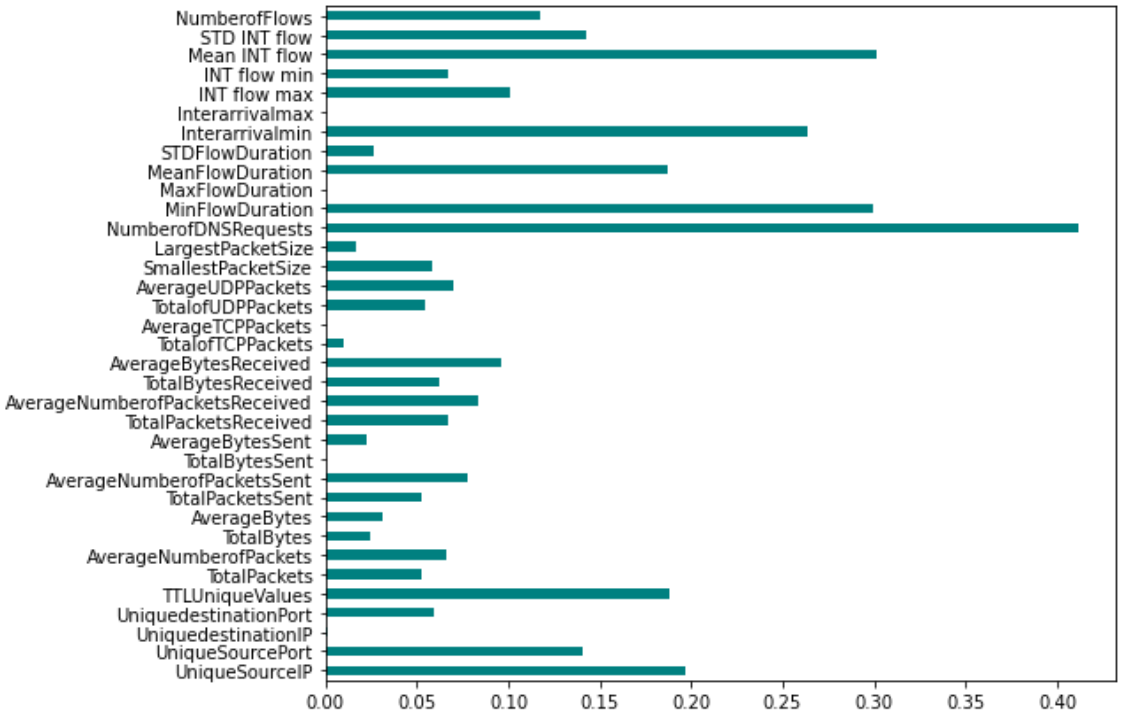


Figure 4. Feature Importance for the Multiclass Host Counting Model

4.1.3 Feature Selection for Binary Number of Host Detection Model

This technique is used in preprocessing to eliminate features that do not affect the ML model. Figure 5 shows the feature importance for binary host count approximation. It shows the relation of each feature with the predicted class. The TTL unique values have the highest coefficient and thus it highly affects the model's prediction. It is shown in the literature that TTL has great evidence on NAT presence and in identifying the number of hosts. Similarly, the average bytes received feature is of secondary importance. This is because the higher the bytes the more devices we have. This study focuses on implementing an ML algorithm on aggregated network traffic features. Thus, if there are more devices more data is sent and thus the total number of bytes received to a specific IP address will be higher. The features that have no impact on the binary host approximation model are Unique Source IP, Largest Packet Size, Interarrival min, and

Int flow min. In the case of NAT, it is normal that the unique source IP will not give evidence of the network size behind a NAT because all active devices behind a NAT have a single shared source IP address.

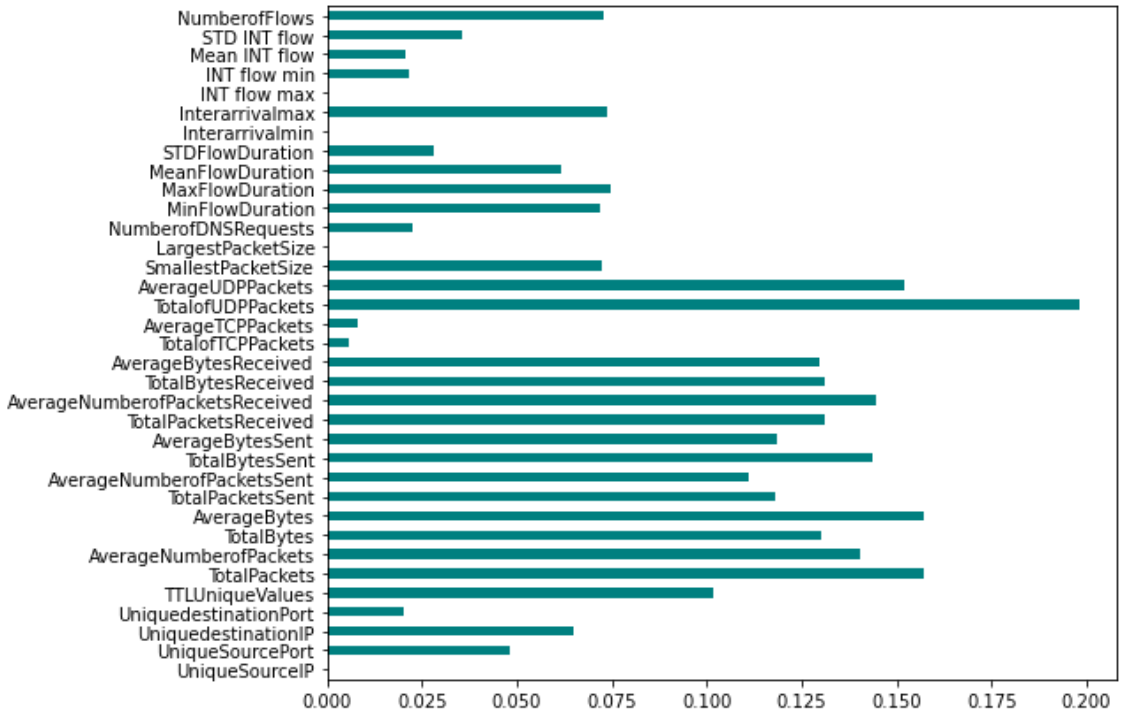


Figure 5. Feature Importance for Network Size Detection

4.2. Hyperparameter Tuning

Each of the algorithms has important parameters which highly affect the model performance. For example, RF requires the number of estimators, which is the number of decision tree classifiers before taking the average predictions, the maximum depth which represents how much the tree will expand to take the decision, and the criterion used to make the decision, entropy for information gain and 'Gini' for Gini impurity. SVM requires the kernel which specifies the mathematical function to calculate the hyperplane and C which is a hyperparameter used to control misclassifying errors. MLP requires the hidden layer sizes which represent the number of neurons in the layer, the activation

function, the solver for weight optimization, and the learning rate. KNN requires the number of neighbors to use by default k-neighbors queries. These parameters are chosen using “GridSearchCV” and cross-validation = 3.

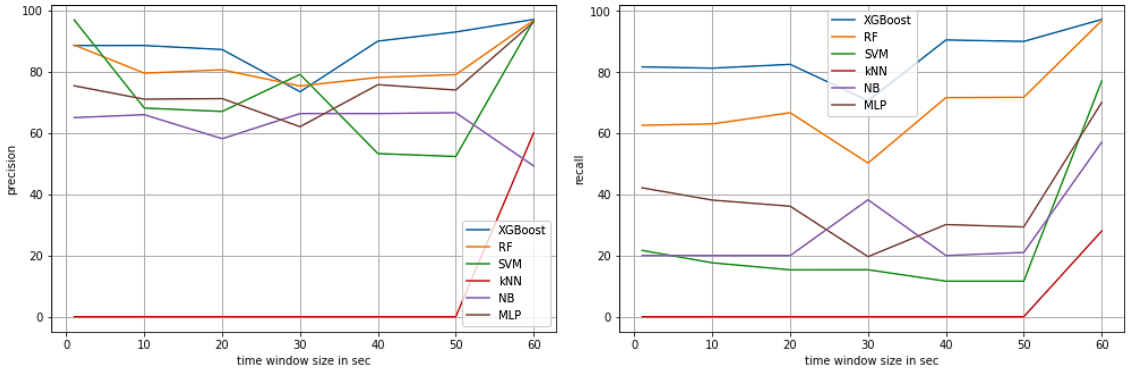
4.3 NAT Detection

4.3.1 Model Selection

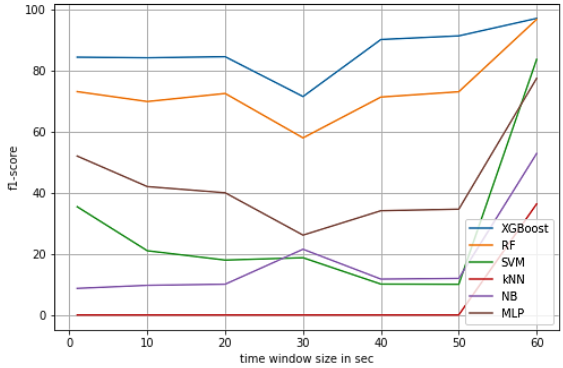
To better evaluate the model performance, cross-validation is used. The data is separated by days using the nested time series cross validation. Starting with 9 days with a 70:30 split between training and test, the test data is fixed and equal to three days. After each fold new data is added, where a day from the test set is moved to the training set and a new day is combined with the test set to test the model’s accuracy on unseen data. Cross validation ends when the 14 days of data are all used 9 for training and 3 for testing. The data is separated into fourteen groups since the dataset represents traffic collected over two weeks. Each group represents a day that contains multiple data samples. The features with low importance as “AverageTCPPackets”, “TotalBytesSent”, and “Interarrivalmax” (refer to Figure 3) are removed.

Multiple experiments were performed to assess the effectiveness of NAT detection using aggregated features from different time windows (duration of time series). Figures 6a, 6b, and 6c show the impact of the machine learning classifier and time window size on detection accuracy. Figure 6a shows that the XGBoost and RF models have high precision at all window sizes, but the highest value is recorded at a time window equal to 60 seconds. This means that both have low true negative rate thus the number of aggregated flows that are misclassified is low. The other machine learning models have lower precision scores than RF and XGBoost in different window sizes. At the time

window = 60 sec, RF, SVM and MLP have lower precision values than XGBoost, but also recorded a high precision at 60 seconds. But since we have an imbalanced dataset, predicting the majority class might lead to high precision, thus we cannot rely on the precision alone to evaluate the model performance. Figure 6b shows the recall of each machine learning classifier at different window sizes. Similarly, the XGBoost and RF classifiers reported higher rates, but all algorithms have a low recall. High precision but low recall means that one of the classes is mostly misclassified. The reported recall scores are below 80% for all window sizes except for RF at 60 sec and XGBoost. The lowest reported recall by XGBoost is about 70% at time window size equal to 30 sec. This means that the XGBoost model also reported high recall values at all window sizes. The overall performance of the system is shown in Figure 6c, which presents the F1 score. Since an ideal machine learning algorithm is that with high precision and recall, yielding a high F1-score, the detection reaches its peak when the aggregation window size is 60 sec. The obtained results show that the window size has a significant impact on the accuracy of NAT detection. For small window sizes, the model fails to find patterns in the features to detect the presence of NAT. XGBoost gives the best results with RF coming as the second-best algorithm. From this experiment we can conclude that increasing the time window sized to 1 min has improved the model detection rate. If we look at RF and XGBoost model, that are the best NAT detection models, we can conclude that the increase in window size enhance the detection rate. To illustrate the reported f1 score at time window = 50 sec is less than that reported at 1 min, similarly for other window sizes.



(a) Precision of the Algorithms (b) Recall of the Algorithms



(c) F1 Score of the Algorithms

Figure 6. Evaluation Metrics of Different ML Classifiers with Different Window Sizes

To compare the results obtained at a time window equal to 1 minute, Table 3 summarizes the F1-score results after evaluating all algorithms. It proves that XGBoost, RF, and SVM outperform the other classification techniques. Besides Naïve Bayes and kNN failed to detect NAT devices.

Table 3: Summary of Performance of all Approaches

Algorithm	**F1 score (%)**
XGBoost	97.09
RF	96.90
SVM	83.65
MLP	77.04
Naïve Bayes	48.45
kNN	35.65

The second experiment is performed to see the model's ability to classify NAT devices after excluding the source IP & port and destination IP & port. Since the best approach for NAT detection on our dataset is XGBoost, this test is done using the XGBoost model. Table 4 shows the F1-score for the XGBoost classifier when varying features. It shows that the model has its best performance when all extracted key features are utilized. Besides, when using statistical features and excluding the source IP & port and destination IP & port the system was still able to detect the presence of NAT effectively with a same F1 score reported in the presence of these features (97%). Thus, in the in the absence of IPs and ports the model depends on other features to detect NAT devices effectively with a high score.

Table 4: Summary of Performance of RF Classifier while Varying Traffic Aggregated Features

Features	**F1 score (%)**
All features except source and destination IPs and ports	97
All features	97.09

Since it is shown in the literature that ML-based algorithms outperform traditional methods and addressed multiple limitations, we applied the OS passive fingerprinting NAT detection technique based on TTL values to the extracted aggregated flows. The performance of this algorithm is compared with the RF classifier results. This method was implemented in [9] and is based on the header TTL values of flows following the approach in [10]. As shown in Table 5, this method accomplished a high F1 score of 92.9%; however, it is limited to specific operating systems. It uses the fact that the NAT router decrements the TTL value, thus it requires previous knowledge of the connected devices and their TTL range to be applied. Our proposed machine learning algorithm accomplished better results when we train our classifier on the extracted traffic features,

as shown in Table 4, where the algorithm reached an F1-score of about 97% when all features are used or when the source and destination IPs and ports are excluded. In addition, our approach is not dependent on specific operating systems, since the classifier learns from the extracted features, and can detect NAT behavior for new unseen data regardless of what is running on the hidden hosts.

Table 5: Results Obtained when Applying TTL Range Method on Dataset

Algorithm	Precision	Recall	F1-score
TTL Range Algorithm	100	86.77	92.92

4.3.2 Generalizability Test

To evaluate the model's generalizability, multiple tests have been implemented. First, the model's performance is tested on wireless NAT dataset after performing obfuscation. The different obfuscation scenarios are presented in Table 6. Then the model's ability to detect NAT devices is tested on the wired NAT dataset, on both original data and after performing obfuscation. Two tests were done. First, we choose to assess the model generalizability on obfuscated and newly collected data in the presence of all important features presented in Figure 3. This test is labeled as test one. Then we choose to remove all features that could be affected after obfuscation, keeping unique source and destination IP and ports with the unique TTL values, to build a model that is not affected by any feature change. This test is labeled as test two.

TABLE 6: Obfuscation Scenarios

Obfuscating size-related features	Uniformly
	Randomly
	By adding dummy bytes
	By padding them
Obfuscating features related to packet counts	Uniformly
	Randomly
Obfuscation timing features as duration and time stamp	Randomly

The results of assessing the model on obfuscated data resulting from changing features as explained in section 3.1.1 are presented in Table 7. Changing a single feature in the data does not highly affect the model's decision. Changing packet-size features uniformly reported 96.33% F1 score; however, randomizing these features reported 74.33% F1 score. Changing the number of packets related features uniformly and randomly and randomly changing the packet sizes have higher effect on the model's decision as the model F1 score dropped by about 20%, 20%, and 22% respectively. By comparing these results with the feature importance presented in Figure 3, it is obvious that the number of packets related features have higher importance, and thus changing these features will affect the model's performance. However, the model is still able to detect the presence of NAT and the lowest F1-score reported when obfuscating features related to a single characteristic is 74.33%.

Sometimes in real network environments, multiple features would be obfuscated together. So, the model is evaluated on the data after performing intended obfuscation on all network features that can be affected by obfuscation. As shown in Table 7, the model fails in predicting NAT in case of uniformly obfuscating data or using randomization. For this reason, a test is done to see the model's performance after removing all features that could be affected by obfuscation. The F1 score reported is 92.61%, it is about 4% less than that reported on the original data before obfuscation and before removing these features. This F1 score is high and shows the model's ability to detect NAT even when fewer features are presented. This reported F1 score is higher than the F1 score reported in test 1 in many obfuscation scenarios (refer to Table 7). Comparing the data presented in test one and test two shows that it is better to remove all features that could be obfuscated when the network data is subjected to obfuscation.

TABLE 7: Results for Tests One and Two on Both Datasets Using the NAT Detection Model

Dataset	1[(1)]	2[(2)]	3[(3)]	4[(4)]
No Obfuscation	97.09	92.61	65.09	68.65
Uniform obfuscation for sizes	96.33	92.61	49.56	68.65
Random obfuscation for sizes	74.33	92.61	44.75	68.65
Obfuscate sizes by adding dummy bytes	93.54	92.61	48.60	68.65
Obfuscate sizes by padding them	96.10	92.61	47.38	68.65
Uniform obfuscation of number of packets	77.10	92.61	51.58	68.65
Random obfuscation of number of packets	76.31	92.61	54.28	68.65
Random obfuscation of duration	96.78	92.61	47.95	68.65
Random obfuscation of timestamp	91.85	92.61	46.29	68.65
Obfuscate all features with uniform obfuscation of sizes and number of packets	56.42	92.61	56.63	68.65
Obfuscate all features with random obfuscation of sizes and number of packets	57.39	92.61	53.25	68.65
Obfuscate all features with random obfuscation of the number of packets and padding sizes	57.08	92.61	55.97	68.65
Obfuscate all features with random obfuscation of the number of packets and add dummy bytes to sizes	57.83	92.61	54.78	68.65
Obfuscate all features with uniform obfuscation of the number of packets and padding sizes	58.31	92.61	53.81	68.65
Obfuscate all features with uniform obfuscation of the number of packets and add dummy bytes to sizes	54.10	92.61	56.20	68.65
average	76.04	92.61	54.41	68.65

The second experiment done to test the model's ability to generalize is repeating both tests one and two on new data collected in a different network environment (wired NAT dataset). Similarly, this data is subjected to intended obfuscation. The data is obfuscated in the same way the previous data features are mutated. Unfortunately, the model did not generalize as it reported an F1-score of 65.09% on the data before

(1) Test 1 performed on wireless NAT dataset
(2) Test 2 performed on wireless NAT dataset
(3) Test 1 performed on wired NAT dataset
(4) Test 2 performed on wired NAT dataset

obfuscation. In test one the model performance in new environment is similar to the model performance on the wireless NAT dataset when multiple features are obfuscated. F1-score dropped more when testing the model on data after obfuscation. In test 2, the f1 score increased by 3.56% however it is still low. Thus, even after removing all obfuscated features to minimize the environment effect the model was not able to detect NAT devices effectively. This means that the model in both tests does not generalize. The NAT detect model failed in a new network environment where there exist different network connections, applications used by users, and devices.

4.3.3 Explainability for NAT Detection

XAI helps in explaining the model's global and local decisions and thus understanding its ability to generalize. By explaining the model local decisions, we can see how a change in feature value can lead to false predictions. Besides by explaining the global decision we can understand the most features that contribute to model's decision and how they are affecting the target variable. This experiment is done by explaining the global decisions by plotting the SHAP feature importance and seeing the key features that are highly affecting the model's decision. Then we used LIME plot to see the model decision explanation for a local data sample.

4.3.3.1 SHAP

This experiment is done to understand how the NAT detection model is taking decisions, why it is not generalizing, and what are the most relevant features. It is important to see if the results taken by the model are realistic or not. SHAP feature importance is applied to see how feature importance is distributed in different datasets. It

computes the mean SHAP value of the most important features that affect the model's decision. Figures 7 and 8 show the SHAP feature importance for test one on wireless NAT data and wired NAT data respectively. Each feature is affected equally in both classes which are shown in the equal distribution of blue and red colors. However, the distribution of feature importance is different, thus in wireless NAT dataset the number of DNS requests, interarrival min, the number of unique contacted destination IP's, the largest packet size, and the mean of interarrival timing between consecutive flows have higher importance. Differently for the wired NAT dataset these features have different importance distribution. For example, the number of DNS requests turned to be the second contributing feature in the wired data. Similarly, the mean of the interarrival timing between flows is the most contributing feature in the mode's decision. This means that the distribution of features in these two datasets is different. Although the TTL unique values might be an indication of the presence of NAT it has less importance in data two. This is because the TTL value is highly dependent on the operating systems and thus if there are two similar OSs behind NAT the TTL unique values will be one and thus leading to incorrect model decisions. Similarly, the number of flows sent is not contributing to the model's decision for wired NAT data. We can illustrate this due to high user activity in the wired NAT data which made the number of flows in case of NAT and end host almost have the same distribution. This means that each network environment has its characteristics and thus the flow duration, the number of packets, and packet sizes are highly dependent on the time the application used is still opened. Thus, the higher the time the application is opened the higher number of packets sent, the larger the total bytes, and the higher the duration. In addition, the interarrival between flows represents the difference between the start time of two consecutive flows. And thus, there is no

guarantee that two different network environments will have the same number of devices, the same operating systems, the same application used, and the same time of activity. This makes the model generalizability harder. The SHAP feature importance shows that aggregated features extracted from the interarrival timing between flows are highly contributing the prediction. These new extracted features are highly related to the NAT presence since in case of NAT, more hosts are active thus more flows are sent and the interarrival timing is smaller. This made the distribution of interarrival timing features different between a NAT and end host, and thus made this feature relevant in case of NAT detection.

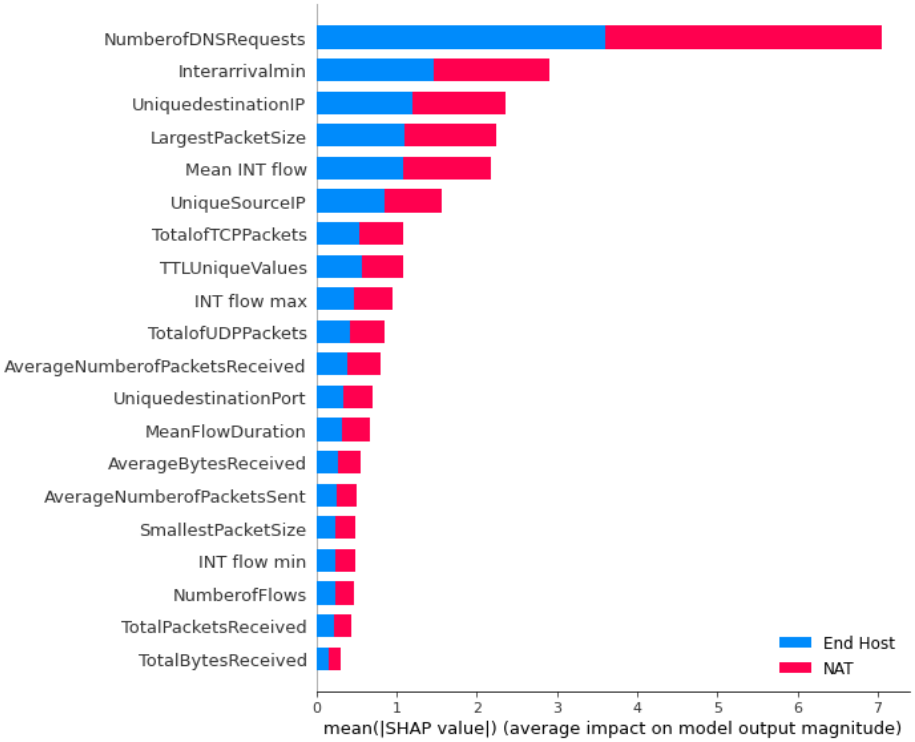


Figure 7. SHAP Mean Absolute Value Plot for the Most Important Features for wireless NAT Dataset

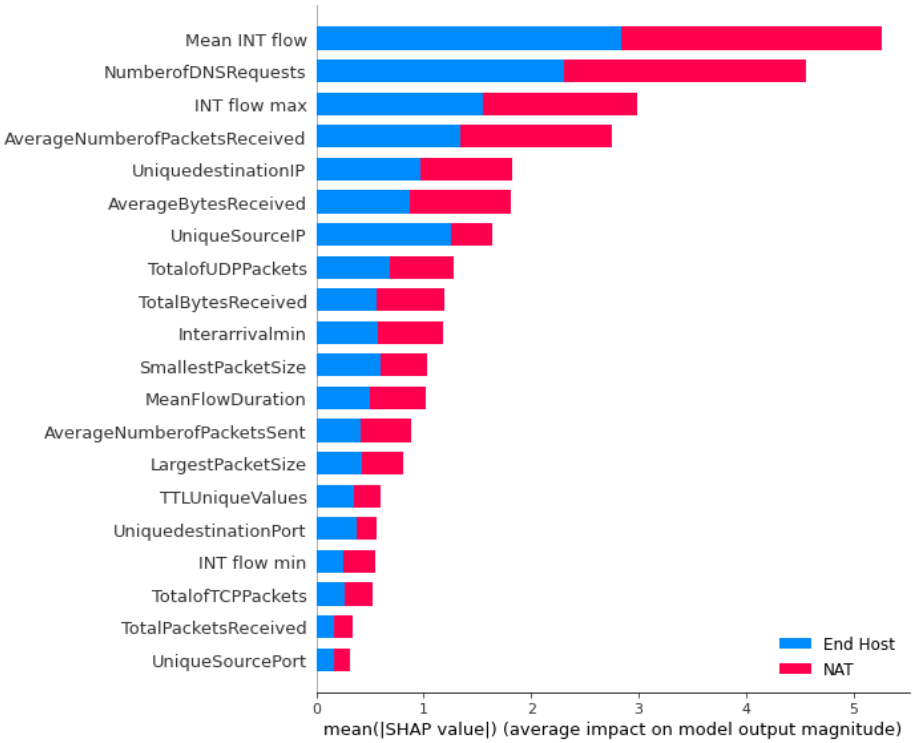


Figure 8. SHAP Mean Absolute Value Plot for the Most Important Features for wired NAT Dataset

Figures 9 and 10 show the feature importance after removing all features that can be changed due to obfuscation for the wireless NAT and wired NAT datasets respectively. The model still has only five features to depend on to decide so any high variation in a specific feature will lead to a higher change in model performance. All features in the model affect equally both decisions, this is seen by equal distribution of classes “End Host” and “NAT” in the figures. Both figures show same distribution of features in both datasets is different. However, Table 7 shows that the model failed to generalize in test two. This means that although the features have same distribution of importance and participation in prediction, it is not necessary that the model generalize. The model’s decision is based on the decision rules learned by the model from the training dataset. Both datasets are collected by varying the devices behind the NAT between 1 and 4, the distribution of TTL unique values is different in both datasets. The difference in operating

systems between both datasets is the reason. In addition, the activity of users is different. Thus, the decision rules learned from wireless data might not be applicable on wired data and the model fails to generalize even after removing all features that could be obfuscated. Because the remaining features in this test are still affected by the change in environment the local explanation in the next section will explain why the model failed to generalize more. This experiment shows that even the feature importance their contribution in prediction is the same, it is not necessary that the model would generalize.

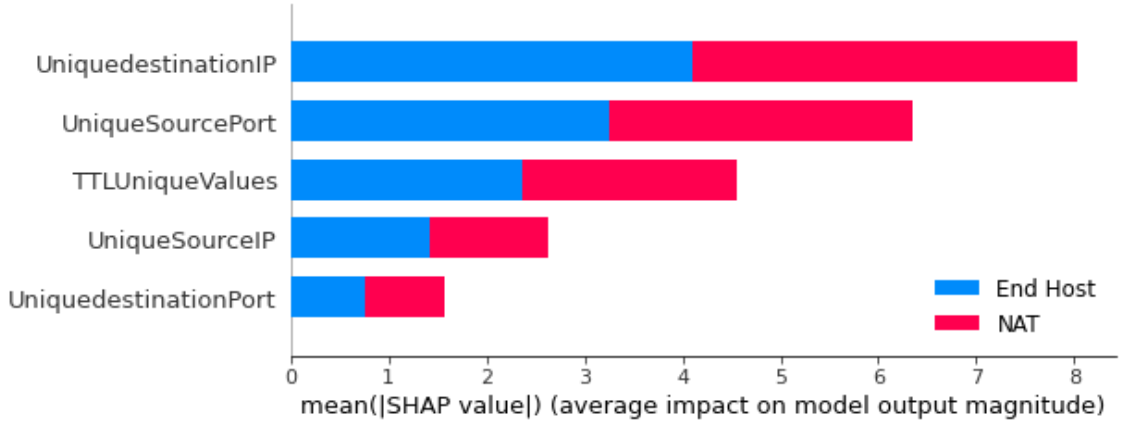


Figure 10. SHAP Mean Absolute Value for the Most Important Features for Wireless NAT Dataset

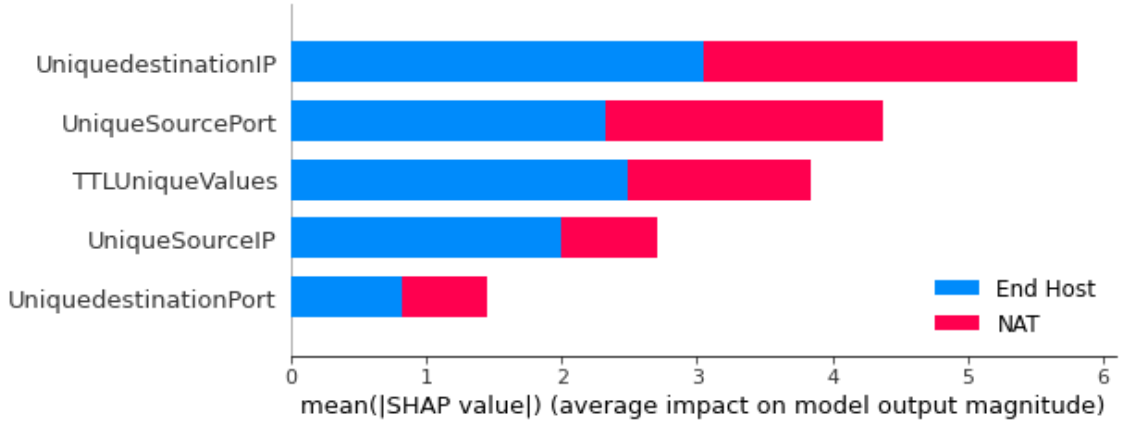


Figure 9. SHAP Mean Absolute Value for the Most Important Features for Wired NAT Dataset

4.3.3.2 LIME

This method is used to explain some local predictions in testing data. For NAT wired and wireless datasets, we used LIME to explain a data sample from test data before obfuscation, after obfuscating a single feature, and when all features are obfuscated. It is shown in Table 7 that in the NAT detection model obfuscating a single feature does not highly affect the model's performance. We applied LIME method to a data sample from the wireless NAT data after obfuscating timestamp features to see how obfuscating a single feature might mislead the detection algorithm. Then because the model fails to predict NAT when multiple features are obfuscated, we chose to show the LIME decision explanation for the same data sample when all features are obfuscated uniformly.

Figure 11 presents the prediction explanation for a wireless data sample before obfuscation. Figure 12 displays the prediction explanation for the same data sample presented in Figure 11 but after obfuscating timestamp features. Figure 13 illustrates the prediction explanation of the same data sample in Figures 11 and 12 but after obfuscating all features uniformly. The three figures show that the actual class for this sample is labeled as zero, i.e., this sample is traffic data outward from a NAT device. Looking at Figure 11, the predicted value by the RF model is 0.0165 which means that the model has a correct prediction. All figures are a two-sided bar plot where the left side is the feature value relation with NAT presence and the right side is the feature value relation to the ordinary host. Figure 11 shows that almost all key features in this sample are related to the NAT presence. After obfuscating timestamp features, the RF NAT detection model predicted value is 0.705 as shown in Figure 12. The RF classifier failed to classify this data sample. The model prediction is above 0.5 because after obfuscation the data sample has a high interarrival time between packets and interarrival between flows. This is

because in the case of an ordinary host only a single device is sending packets, however in the case of NAT multiple devices are active. More devices mean more packets and flows are sent which will decrease the interarrival time between packets and flows. In Figure 13 the model prediction is 0.998 which means that the model predicts this sample as an ordinary host. The main reason behind this false prediction is the high value of the minimum interarrival time between packets, a high number of DNS requests, and the average bytes received is higher. Thus, obfuscating most of the features will lead to a high change in the dataset which will cause false predictions. Even using random or uniform obfuscation of features, with the change of values of more features the model will start relating these values to the presence or absence of NAT based on the decision rules added by the model after being trained on the original form of data.

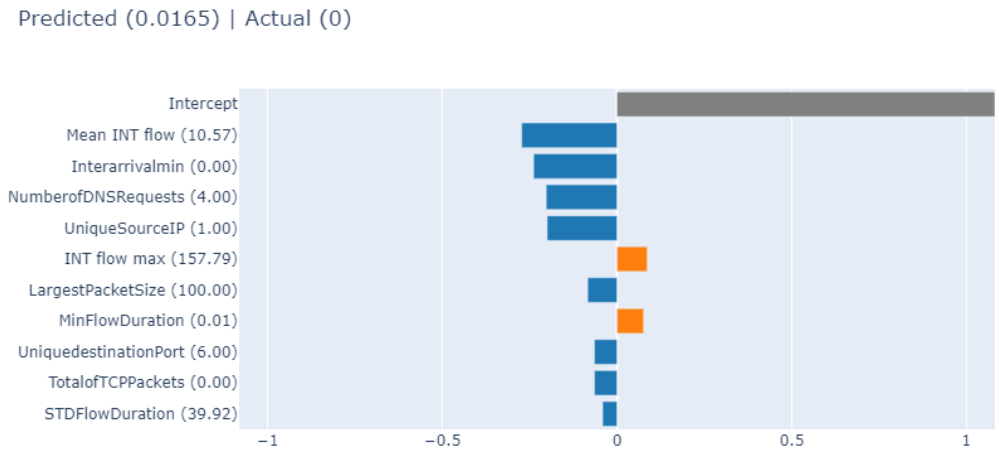


Figure 11. LIME Explanation for a Wireless NAT Data Sample Before Obfuscation

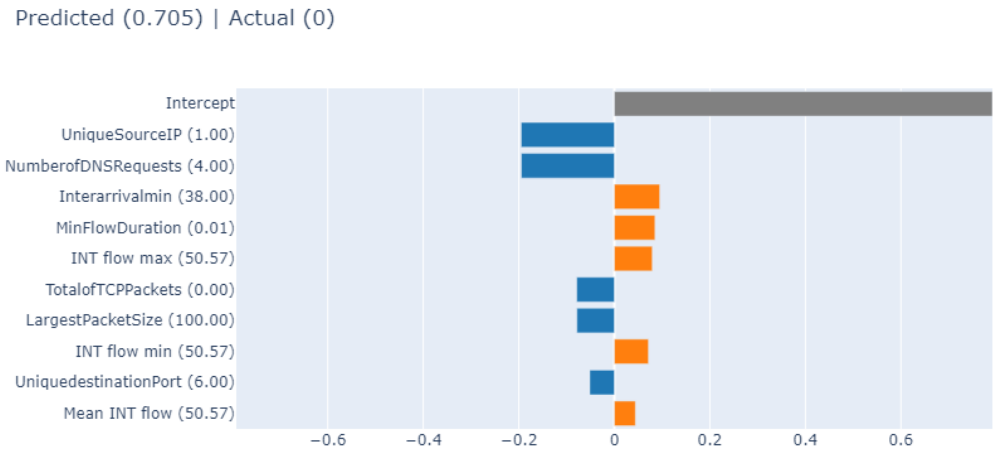


Figure 12. LIME Explanation for a Wireless Data Sample after Obfuscating a Single Feature

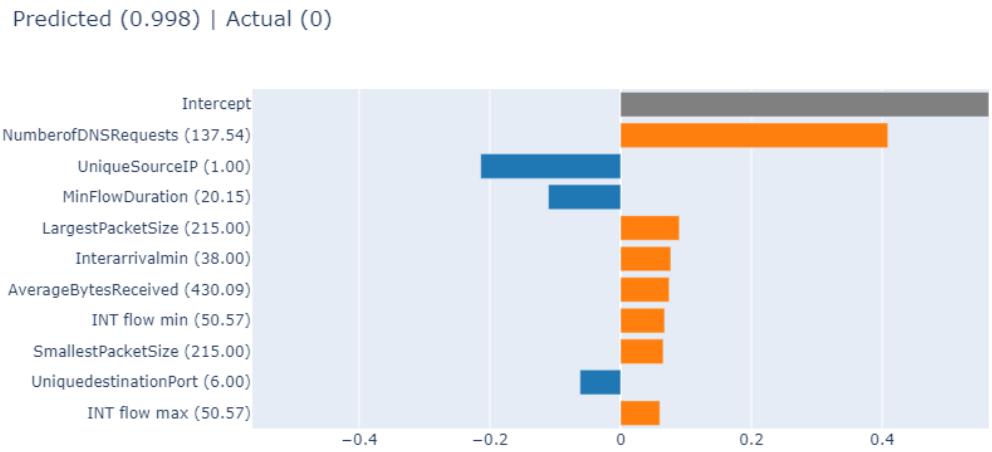
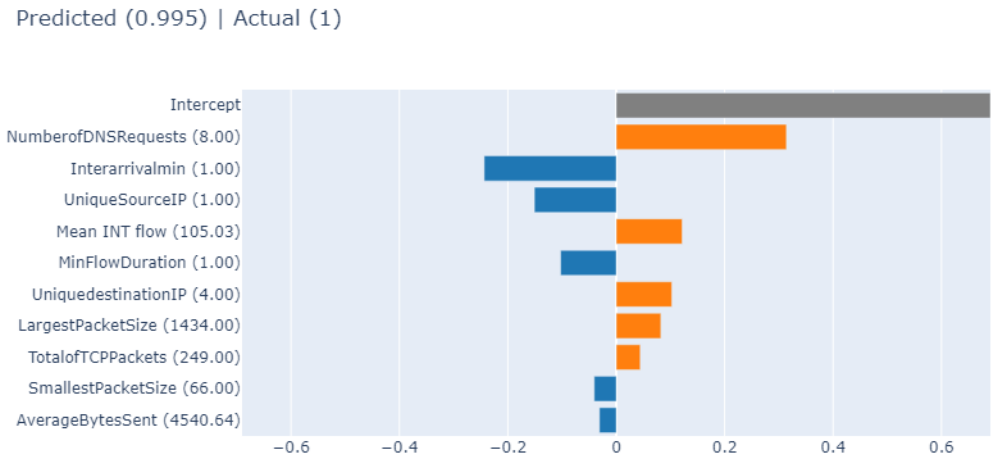


Figure 13. LIME Explanation for a Wireless Data Sample after Obfuscating Multiple Features

Figures 14a and 14b show two samples from wired NAT dataset, one is classified correctly and the other is not classified correctly. In a new dataset, the distribution of features is different thus the model will start comparing these features to the decision rules. Both data samples have predicted as data coming out from end host because the RF model learned when the interarrival between packets is high, the number of packets is low, and there are less unique contacted destination IP addresses it is more likely that the data is coming out from a end host.



(a) First Data Sample

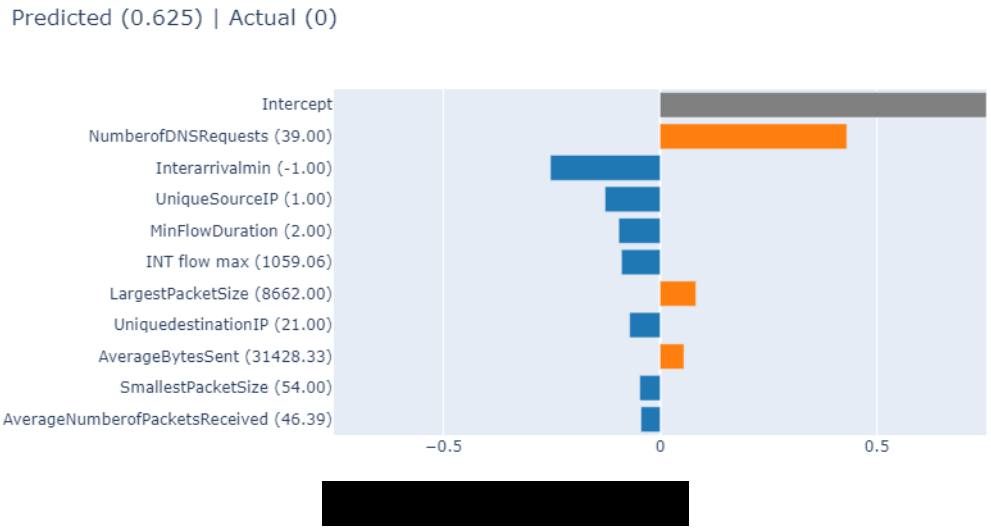


Figure 14. LIME Explanation for Two Data Samples Drawn from wired NAT Dataset

For test two, after removing all features that could be affected by obfuscation, LIME is applied for the wireless and wired NAT datasets before obfuscation. This experiment aims to study why the model failed to generalize even after removing all features that could be affected by obfuscation. Figures 15 and 16 show the LIME prediction explanation for a random sample taken from datasets 1 and 2. The samples represent the aggregated flow out from an ordinary host. The model predicts the sample in Figure 15 correctly but fails to predict the sample in Figure 16 which is drawn from a new environment. It seems that after removing all obfuscated features, the remaining

features values represents an ordinary host. The data sample in figure 16 resembles the data coming out from NAT device based on what the model have learned from the wireless NAT data. From the two figures we can see that a single source IP address is evidence for NAT presence. Besides the small values of destination IPs and ports are evidence for ordinary host. This means that, from training data, the model learned that in the presence of NAT device, the contacted IPs and ports are higher, and this is because there is more than one device active behind a unique single source IP address. However, in Figure 16 the data resembles the data coming out from NAT which leads to false predictions.

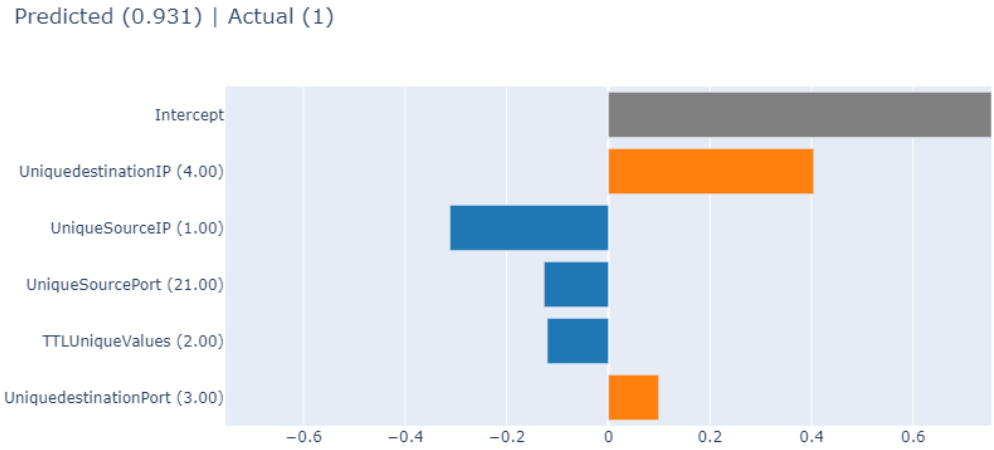


Figure 15. LIME Explanation for a Data Sample Drawn from Test Sample of Data in Wireless NAT Dataset

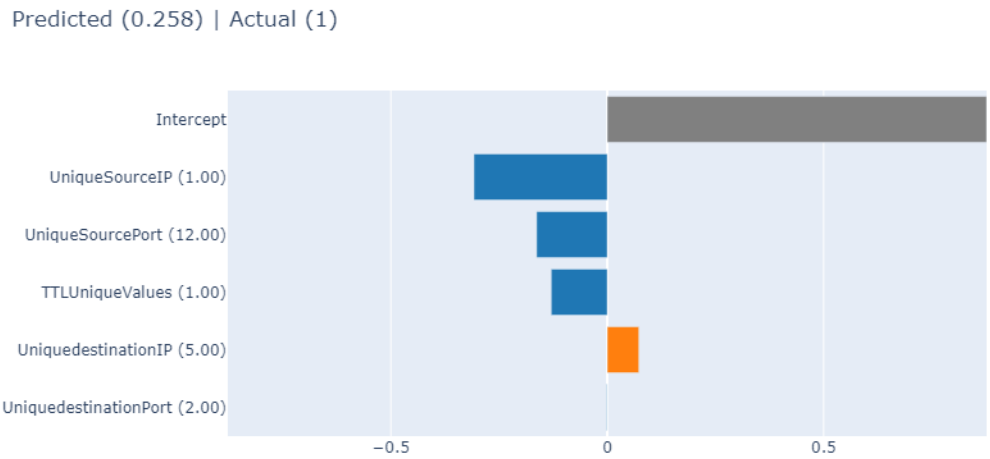


Figure 16. LIME Explanation for Data Sample Drawn from Wired NAT Dataset

4.3.4 Transfer Learning

From previous experiments, it was obvious that it is hard to implement a model that will generalize in all network environments. Thus, transfer learning is used to transfer knowledge from the implementation to enhance the prediction in an unfamiliar environment. For this purpose, a small portion of wired NAT dataset (519 samples) is taken to make locally optimal modifications to each tree structure in the XGBoost model and thus has better performance in a new network environment.

The results for testing the transferred model are presented in Table 8. Transfer learning is applied on both tests one and two. This table shows that transfer learning improves the model performance, where the performance on wired NAT dataset improved by 13% without obfuscation. The F1 score also increased by about 3% after transfer learning while removing all obfuscated features from wired NAT dataset. The F1 score has an average of 72.72% in test one. This average is higher than the reported F1 score on this data after removing all aggregated features that could be changed due to obfuscation. This means that the transferred model performs better in test one and in case of the wired data it is better to keep all features even if there is obfuscation.

Table 8 also show the testing results after retraining the XGBoost NAT detect model on the wired NAT data. The results are presented on column "$5^{(5)}$". The F1 score reported is 77.13 before obfuscation. The performance of the retrained model is almost similar to the performance of the transferred model. This means that the transferred model is optimized and able to work in different network environments with the presence of obfuscation. Yet the results of the transferred model are not too efficient. For this reason, we applied transfer learning for RF because it has a high performance on the wireless NAT dataset as shown in Table 3. To verify the performance of the transferred model on

new data and if it can generalize to new network environments, we tested it on the wireless NAT dataset. The transferred model yields high performance on both datasets 1 and 2. Testing it on wireless NAT dataset the F1 score dropped by 0.28% when there is no obfuscation and 0.15% after removing all obfuscated features.

The testing results for the RF classifier after performing transfer learning are presented in Table 9. This RF transferred model is more efficient on wired NAT dataset than the XGBoost transferred model. The RF transferred model performance on wired NAT dataset improved where the reported F1 score without obfuscation is 88.31% and 83.28 without obfuscation. These reported f1 scores are higher than the F1 scores of XGBoost transferred model on wired NAT data by 10.31% without obfuscation (test 1) and 5.15% on test two. Similar to the original model, the transferred model is not highly affected when changing features related to one network characteristic. However, the performance drops when all features are obfuscated thus, we conclude that when there is obfuscated data it is better to use the model after removing all features that could be obfuscated. To verify that this model can generalize in case of obfuscated data collected in new network environment, we tested it on the wireless NAT dataset. The transferred model yields high performance on both datasets 1 and 2. Testing it on wireless NAT dataset the F1 score dropped by 2.5% when there is no obfuscation and 2.78% after removing all obfuscated features. Thus, transfer learning improved the model performance for new data and enhances model generalizability.

Table 8: Results for Tests One and Two on Both Datasets using XGBoost NAT Detection Model after Transfer Learning

Dataset	1[(1)]	2[(2)]	3[(3)]	4[(4)]	5[(5)]
No Obfuscation	96.81	92.46	78.00	71.50	77.13
Uniform obfuscation for sizes	95.89	92.46	72.90	71.50	77.38
Random obfuscation for sizes	80.46	92.46	70.74	71.50	72.27
Obfuscate sizes by adding dummy bytes	95.26	92.46	75.22	71.50	76.81
Obfuscate sizes by padding them	96.08	92.46	77.41	71.50	76.38
Uniform obfuscation of the number of packets	77.11	92.46	72.13	71.50	89.48
Random obfuscation of number of packets	82.49	92.46	71.67	71.50	72.59
Random obfuscation of duration	96.83	92.46	73.21	71.50	94.38
Random obfuscation of timestamp	88.04	92.46	73.46	71.50	99.64
Obfuscate all features with uniform obfuscation of sizes and number of packets	61.46	92.46	69.18	71.50	72.44
Obfuscate all features with random obfuscation of sizes and number of packets	59.74	92.46	73.41	71.50	68.05
Obfuscate all features with random obfuscation of the number of packets and padding sizes	60.43	92.46	70.92	71.50	70.76
Obfuscate all features with random obfuscation of the number of packets and add dummy bytes to sizes	61.85	92.46	70.47	71.50	70.33
Obfuscate all features with uniform obfuscation of the number of packets and padding sizes	55.80	92.46	70.79	71.50	71.43
Obfuscate all features with uniform obfuscation of the number of packets and add dummy bytes to sizes	56.38	92.46	71.32	71.50	71.41
average	77.64	92.46	72.72	71.50	77.36

(1) Test 1 performed on wireless NAT dataset
(2) Test 2 performed on wireless NAT dataset
(3) Test 1 performed on wired NAT dataset
(4) Test 2 performed on wired NAT dataset
(5) Retrain the XGBoost model on wired NAT using test 1

TABLE 9: Results for Tests One and Two on Both Datasets using RF NAT Detection Model after Transfer Learning

Dataset	1[(1)]	2[(2)]	3[(3)]	4[(4)]
No Obfuscation	94.17	89.90	88.31	83.23
Uniform obfuscation for sizes	89.97	89.90	87.65	83.23
Random obfuscation for sizes	77.16	89.90	84.13	83.23
Obfuscate sizes by adding dummy bytes	92.37	89.90	87.73	83.23
Obfuscate sizes by padding them	91.28	89.90	87.06	83.23
Uniform obfuscation of the number of packets	82.96	89.90	83.48	83.23
Random obfuscation of number of packets	85.96	89.90	81.78	83.23
Random obfuscation of duration	94.22	89.90	87.93	83.23
Random obfuscation of timestamp	71.75	89.90	83.38	83.23
Obfuscate all features with uniform obfuscation of sizes and number of packets	42.69	89.90	81.70	83.23
Obfuscate all features with random obfuscation of sizes and number of packets	36.24	89.90	70.48	83.23
Obfuscate all features with random obfuscation of the number of packets and padding sizes	36.33	89.90	70.33	83.23
Obfuscate all features with random obfuscation of the number of packets and add dummy bytes to sizes	36.33	89.90	69.61	83.23
Obfuscate all features with uniform obfuscation of the number of packets and padding sizes	55.80	89.90	74.35	83.23
Obfuscate all features with uniform obfuscation of the number of packets and add dummy bytes to sizes	57.51	89.90	69.56	82.23
average	69.64	89.90	80.49	83.23

4.4 Host Number Identification

4.4.1 Model selection

We can treat the host counting problem either as a multiclass classification problem or a binary classification problem. Both wireless and wired NAT datasets have four classes. The classes represent the number of devices connected behind the NAT

(1) Test 1 performed on the dataset presented in section 3.1.1
(2) Test 2 performed on the dataset presented in section 3.1.1
(3) Test 1 performed on the dataset presented in section 3.1.2
(4) Test 2 performed on the dataset presented in section 3.1.2

within the aggregated time window. The number of devices is varied from one up to four based on the test done. Previous studies have implemented multiple algorithms to either approximate or find the exact number of hosts. Yet these studies are still limited to certain operating systems, or the maximum number of hosts hidden. In this thesis, the aim is to find a model that can detect the hidden network size without being limited to a specific type of OS or a maximum number of hosts in training data. For the second model, the multiclass problem is transformed into a binary classification problem. This will improve the classification, make more realistic predictions, and will not be limited to only four devices. The model will predict if the number of devices is few (<= 2) or many (>2).

The proposed approach consists of a cascade of classifiers where the data coming out from the NAT detection algorithm is fed to a machine learning classifier to detect the size of the network. Similarly, nested time series cross-validation is used, and multiple machine learning algorithms are applied in case of multiclass and binary classification to find the best algorithm that can detect the size of the network. For this multiclass classification problem, the F1 score resulting after applying the machine learning algorithms is presented in Figure 17. In this figure, the red bars represent the F1 scores of the models proposed to detect the exact number of active devices behind the NAT. The green bars represent the F1 scores obtained by the models proposed to approximate the size of the network. In multiclass classification the RF classifier has the highest F1 score, whereas in the binary classification problem the XGBoost outperforms other algorithms. The binary classification model has higher F1 score than the multiclass in all implemented ML algorithms. This means that the approximate host count model gives more accurate results than the exact host count model. Besides, the highest F1 score reported by the multiclass model is about 62% which is considered low. Yet the binary model reported a

high F1 score which is about 90%. In this case, it is better to use the binary model because we are using data labeled with four classes and thus a multiclass model will not generalize in case there exist a higher number of devices.

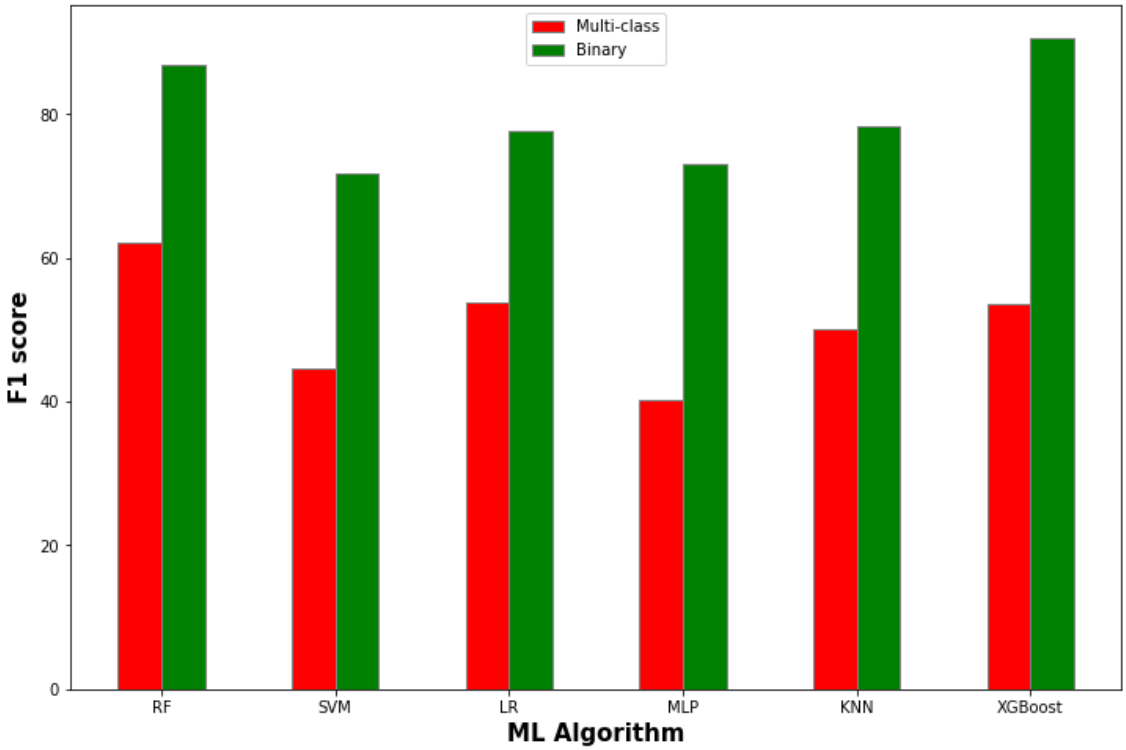


Figure 17. F1 Scores for Different Machine Learning Algorithms for Binary and Multiclass Host Counting Models

Although the multiclass problem yields a low F1 score, it is important in multiclass classification problems to see the confusion matrix in order to understand the model performance in each class. Figure 18 shows the confusion matrix of the Random Forest model. The confusion matrix shows that the model is more likely to make wrong predictions between near classes. The model shows that about 30% of the data labeled as two, i.e., there are two active devices behind NAT, is classified as one by the model. Similarly for the other two classes about 25% of the data belonging to class 4 is classified as 3 and 37% belonging to three is classified as four. Thus, a high percentage of near classes is misclassified which makes the F1 score for the counting model, i.e., multiclass model, is low. Although the model has a low F1 score, it has a low margin of error where

it is making wrong predictions between near classes. In this case, the machine learning algorithm is not able to define fingerprints for near classes by examining the aggregated flow generated by the hosts. The performance of the model is expected because not all devices behind a NAT need to be active at the same time, or their activity starts together. Thus, even though in this time window there are two devices connected behind the NAT it might be that the first device activity starts at the beginning of this time window whereas the second device started almost at the end. This leads to a small number of flows coming out from device two which made the data resemble the case where there exists only one hidden device. Since the model fails to find patterns to differentiate between near classes, it is hard to predict the exact number of hosts in the presence of NAT.

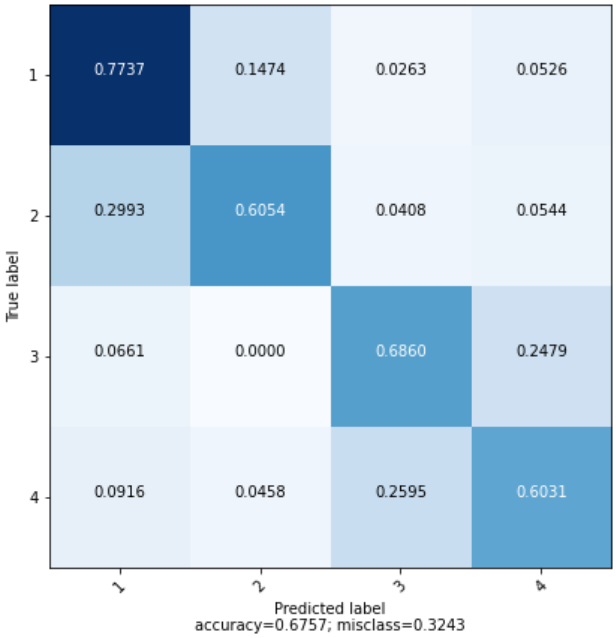


Figure 18. Confusion Matrix to Explain Predictions
in Multiclass Host Classification Model

4.4.2 Generalizability Test

Since the XGBoost classifier outperformed all other machine learning algorithms in the binary host counting experiments, we have evaluated its immunity to obfuscation and change in environment. The model generalization test is done on both wireless NAT

dataset after obfuscation and wired NAT dataset with and without obfuscation. This test is performed by conducting the network size approximation experiment on the training data and then using the unseen data as a testing dataset to study the generalization. We implement the generalization test on the binary host counting model because it can detect the network size more easily. Similar to NAT detection, we test the XGBoost model on obfuscated data in the presence of obfuscated features (test one) and after removing all obfuscated features (test two).

We started with assessing the model's ability to adapt properly to obfuscated data drawn from the same distribution of the dataset used in training the model. This experiment is done using the obfuscated versions of wireless dataset. The F1 scores obtained are presented in Table 10. The table shows that the model performance when there is only one feature obfuscated is not highly affected except for random obfuscation of packet sizes. The lowest scores reported for other single feature obfuscated datasets is 70.86% and 52.70% which is about 20% and 58% less than the F1 score reported on the data before obfuscation. This score is reported when obfuscating the features that are related to number of packets and packet sizes randomly. This means that the random obfuscation of packets and their sizes is highly affecting on the model's decision. However, the model fails to approximate the number of devices when multiple features are obfuscated, and thus removing all obfuscated features in this case leads to less false predictions. Test two resulted in an F1 score equal to 76.81 which is higher than multiple scores reported in different obfuscation scenarios especially when multiple features are obfuscated. The host counting model was able to generalize when the testing data is obtained from the same distribution of the training data, even after obfuscation.

Table 10: Results for Tests one and two on Both Datasets Using the Host Counting Model

Dataset	1[1]	2[2]	3[3]	4[4]
No Obfuscation	90.63	76.81	64.52	82.58
Uniform obfuscation for sizes	80.52	76.81	66.50	82.58
Random obfuscation for sizes	52.70	76.81	38.55	82.58
Obfuscate sizes by adding dummy bytes	84.48	76.81	54.40	82.58
Obfuscate sizes by padding them	81.12	76.81	63.33	82.58
Uniform obfuscation of number of packets	85.36	76.81	63.62	82.58
Random obfuscation of number of packets	70.86	76.81	55.33	82.58
Random obfuscation of duration	86.03	76.81	61.63	82.58
Random obfuscation of timestamp	89.73	76.81	64.52	82.58
Obfuscate all features with uniform obfuscation of sizes and number of packets	55.19	76.81	55.98	82.58
Obfuscate all features with random obfuscation of sizes and number of packets	74.02	76.81	55.25	82.58
Obfuscate all features with random obfuscation of the number of packets and padding sizes	71.38	76.81	52.02	82.58
Obfuscate all features with random obfuscation of the number of packets and add dummy bytes to sizes	74.87	76.81	51.96	82.58
Obfuscate all features with uniform obfuscation of the number of packets and padding sizes	62.80	76.81	63.83	82.58
Obfuscate all features with uniform obfuscation of the number of packets and add dummy bytes to sizes	71.12	76.81	61.53	82.58
average	75.39	76.81	58.20	82.58

We performed another experiment to test the model's ability to generalize on data captured in a different network environment. This experiment is done on wired NAT dataset before and after obfuscation. Although the model reported high results on wireless NAT dataset, it fails to approximate devices in the wired NAT dataset. Table 10 shows a low F1 score reported by the XGBoost model in both tests on original and obfuscated data. The highest F1 score in this experiment is reported in test one is 64.52% and resulted

[1] Test 1 performed on wireless NAT dataset
[2] Test 2 performed on wireless NAT dataset
[3] Test 1 performed on wired NAT dataset
[4] Test 2 performed on wired NAT dataset

before obfuscating this data. This score is still low and thus the model is not able to properly detect the network size behind a NAT. However, in test two the F1 score increased to reach 82.58%. Removing the features that could be changed due to obfuscation in the counting model improved the model's performance on new data even after obfuscation. As in NAT detection, the model built from wireless NAT dataset cannot correctly predict the size of the network on data captured in a new network environment in test one. The counting model was able to generalize and detect the network size effectively in case of obfuscation and environment change in test two.

4.4.3 Explainability of Host Counting

In host profiling experiments XAI is used to explain the model's ability to generalize and the low performance of multiclass model. SHAP mean absolute value for key features is plotted to see how the model is taking decisions and how the features are related to the class. SHAP is used to explain the reason the model fails to generalize and why the multiclass model has such a low performance. LIME is used in order to plot samples from datasets and see what makes the model generalizability harder.

4.4.3.1 SHAP

The host counting model fails to generalize when testing it on unseen data for test one. To explain the model's ability to generalize, XAI is applied. To generate global explanation the absolute mean SHAP value for most key features is calculated. This value is plotted in a bar graph to show the SHAP feature importance. SHAP feature importance is an explainable method that illustrates how the proposed model is taking decisions. Figures 19 and 20 present the feature importance of both wireless and wired NAT datasets

respectively for test one. The red color represents “Small NAT” where the network behind NAT device contains more than two devices. The blue color represents “Large NAT” where the number of devices behind the NAT is less than or equal to 2. The distribution of red and blue colors is equal in both figures for all features. This means that both classes use these features equally. Figures 19 and 20 show that TTL unique values has the highest absolute mean SHAP value and thus it has the highest effect on the model’s decision. Thus, any change in TTL unique values might lead to false predictions. The average bytes received is the second important feature for wireless NAT data. However, in wired NAT dataset, looking at Figure 20 we can find that the average bytes received is not highly contributing to the prediction. The two datasets are different, they have different distribution of features and the fingerprints gained from the training data cannot be applied to the test data to make correct decisions. Figure 20 shows that the number of flows sent during the time aggregation widow size is an important feature that is contributing to the decision. In addition, the minimum interarrival time between flows has a high contribution in prediction. This means that these two new extracted features are beneficial in network size detection. Not only these two features are relevant but also aggregating features enhances the prediction rate because it increases the chance that more hosts would be active.

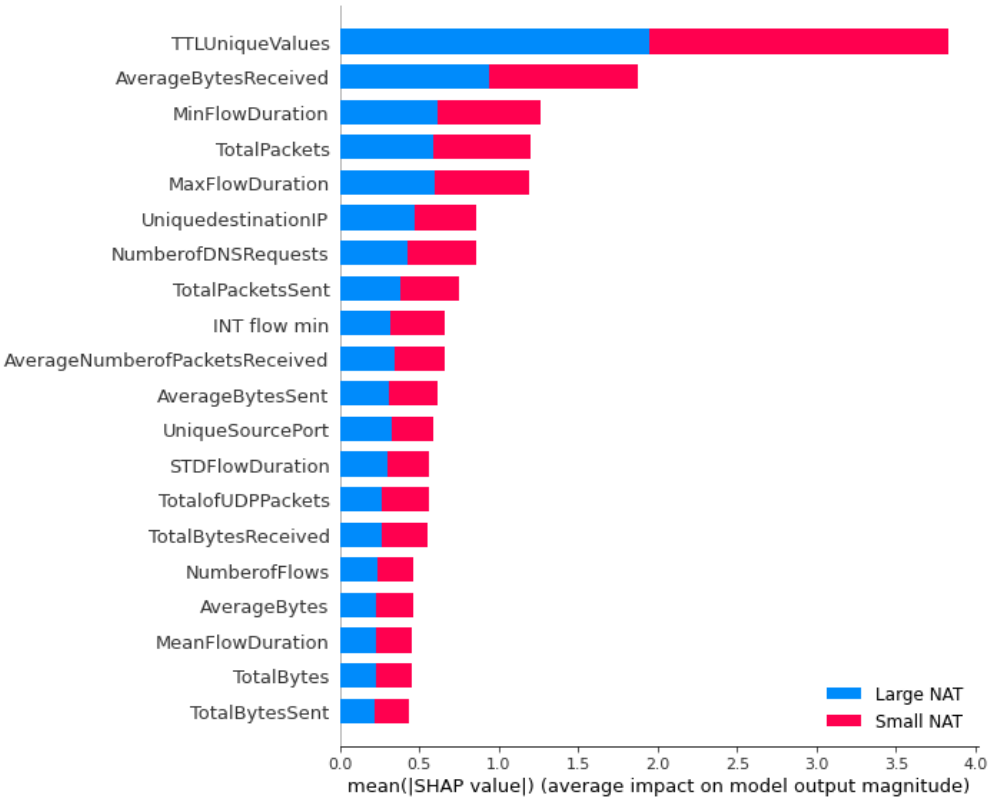


Figure 20. SHAP Mean Absolute Value for the Most Important Features for Wireless NAT Dataset

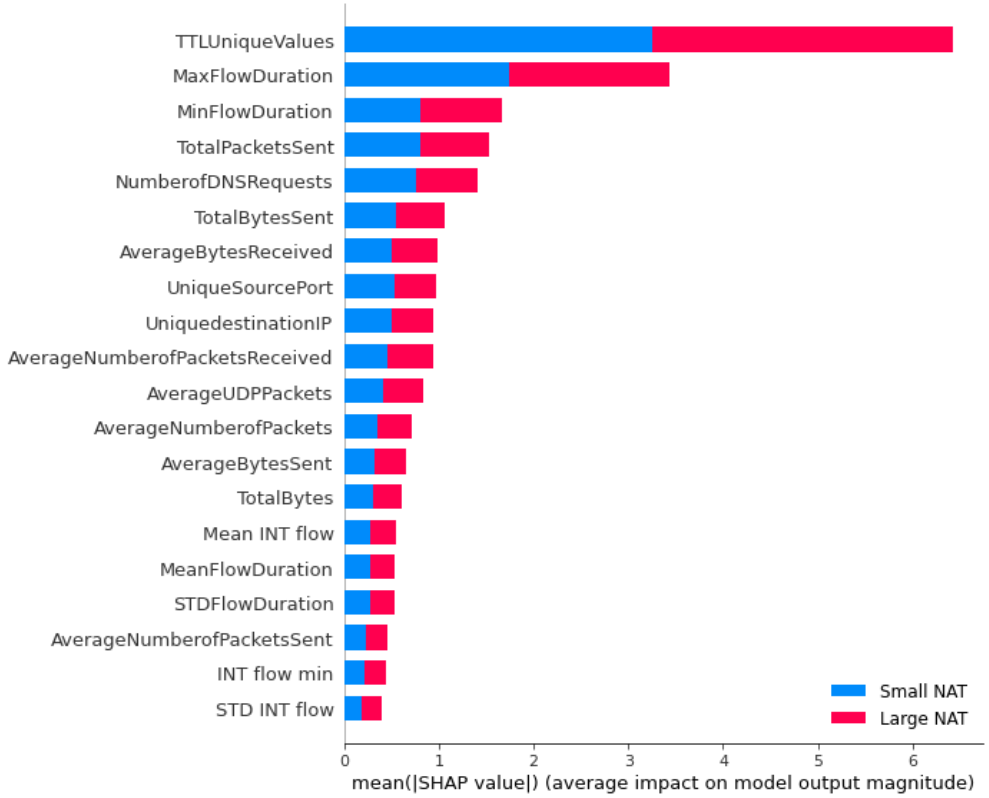


Figure 19. SHAP Mean Absolute Value for the Most Important Features for Wired NAT Dataset

The second experiment was done to see the SHAP feature importance for both datasets when removing all features that can be affected by obfuscation. Yet the generalizability test results show that even in this test the proposed model is able to generalize in a new network environment. Figures 21 and 22 show the SHAP feature importance for the host counting model in test two. For wireless NAT dataset, looking at Figure 21 we can see that the TTL unique values are of the highest importance followed by the number of unique destination IPs. The features have equal effects on both classes. Similarly for the wired NAT dataset the feature distribution is similar to wireless dataset after removing all aggregated features that could be changed due to obfuscation. Although these remaining features are highly dependent on environment the model still able to generalize. To illustrate, the TTL unique values is the most important feature which has the highest SHAP mean absolute value, so it is the most contributing feature in the model's decision. In case of small NAT, the maximum number of hidden hosts is set to 2 in this experiment thus whatever the environment is the values will be either 1 if the two devices have same OS or 2 if the devices are from different OS. This made the decision rules learned from wireless NAT data applicable to the wired NAT data.

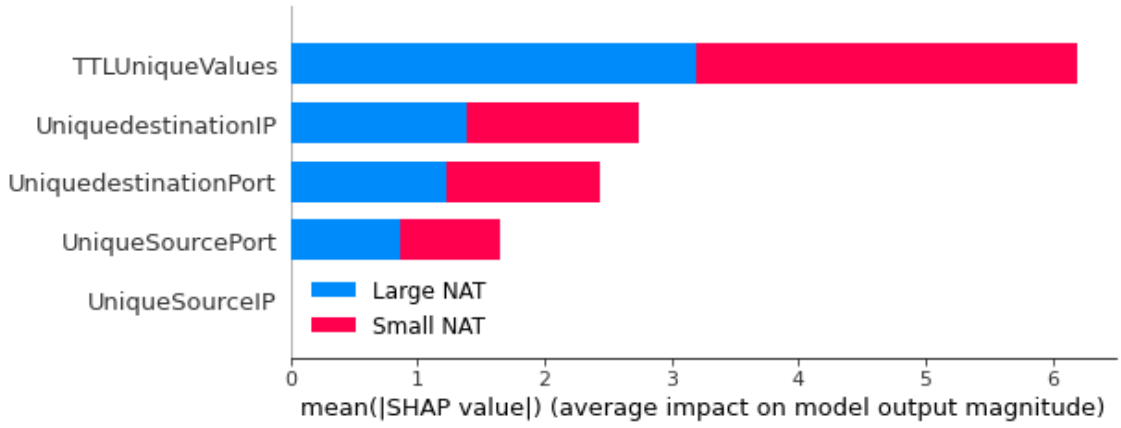


Figure 21. SHAP Mean Absolute Value for the Most Important Features for Wireless NAT Dataset

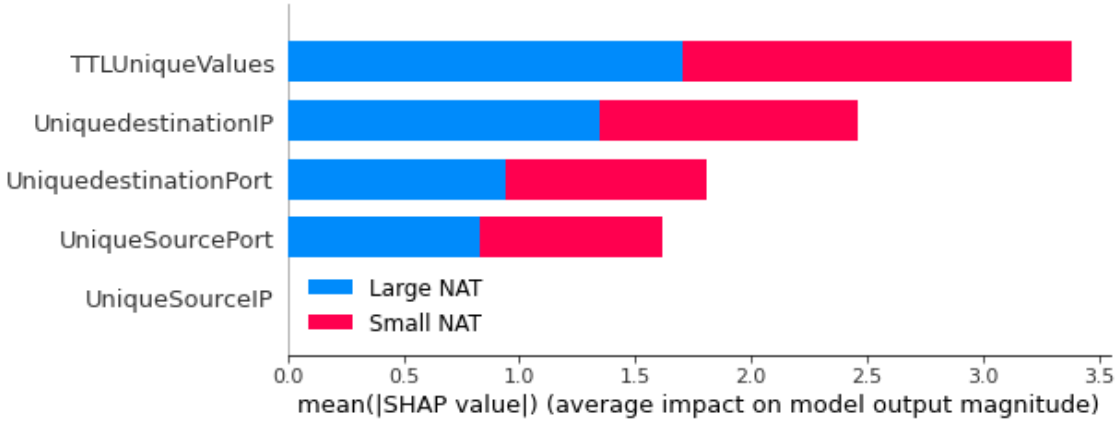


Figure 22. SHAP Mean Absolute Value for the Most Important Features for Wired NAT Dataset

Explainable AI is also used to understand why the multiclass host counting model was not able to detect the exact number of hosts. Figure 23 shows the mean SHAP values for the key features used by the model. The color bar which represents each class for each feature is not distributed evenly across the classes. This means that TTL unique values gives evidence when there exist 1 or 4 hosts connected since the color range for these two classes is larger. So, the model can differentiate between class 1 and 4 easily. However, it has minimum effect on detecting the other two classes. This made a bias between class near classes. To illustrate wired NAT dataset contains four devices with two operating systems, Linux and windows. This made the TTL unique values equals either 1 or 2. This led to the bias where when there are 2 devices with the same OS behind NAT the TTL unique value will be one and the model will predict this case as there is only a single device behind the NAT. Some features like average number of packets received are affecting equally all classes.

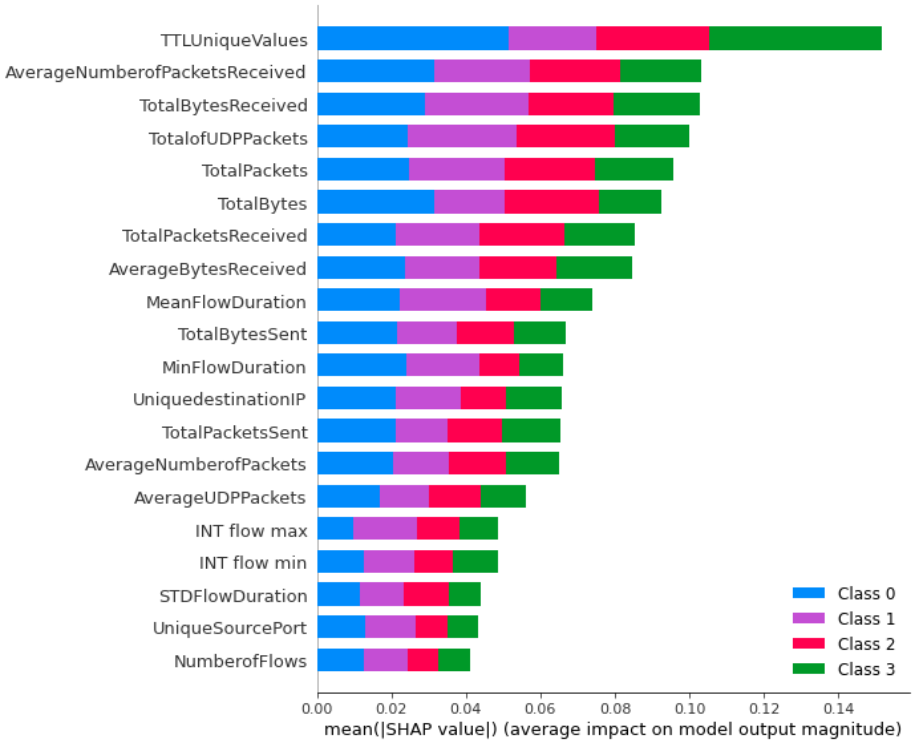


Figure 23. SHAP Mean Absolute Value Plot for the Most Important Features in wireless NAT Dataset

4.4.3.2 LIME

Using LIME, we plotted the binary classifier decision explanation for data samples taken from wireless and wired NAT datasets. This plot represents how each feature value is affecting the model prediction. Figures 24, 25, and 26 show the LIME prediction explanation for the same data sample for wireless NAT dataset before obfuscation, after obfuscating timestamp features, and when obfuscating all features uniformly. Figures 27 and 28 correspond to the LIME explanation for the wired NAT dataset. From the figures we can conclude that when the total TTL unique value is two, there is evidence for a large network behind a NAT. When the number of TTL unique values is one, there is evidence for a small network hidden behind a NAT. This cannot be generalized since for a network that consists of many devices with the same operating system, the total TTL values will be one, but the network hidden is large. Yet the figures

show that there are multiple important features that can help the model in making a correct decision. In Figures 24 and 25 the model predicts the class correctly although the timestamp features are obfuscated in Figure 25. Yet when multiple features are obfuscated, the models fail to find a pattern in the data which will be predicted correctly when applying the decision rule. For the new environment the model was able to predict the sample when there are few devices behind the NAT (<=2) but when the number of devices increases the model fails. To illustrate, there are some values that can be considered as a small network based on the decision rules of the RF classifier. To illustrate, Figure 26 shows that when the total and average TCP packets, and average bytes received are low the model is able to predict that there are few devices behind a NAT than for a large network.

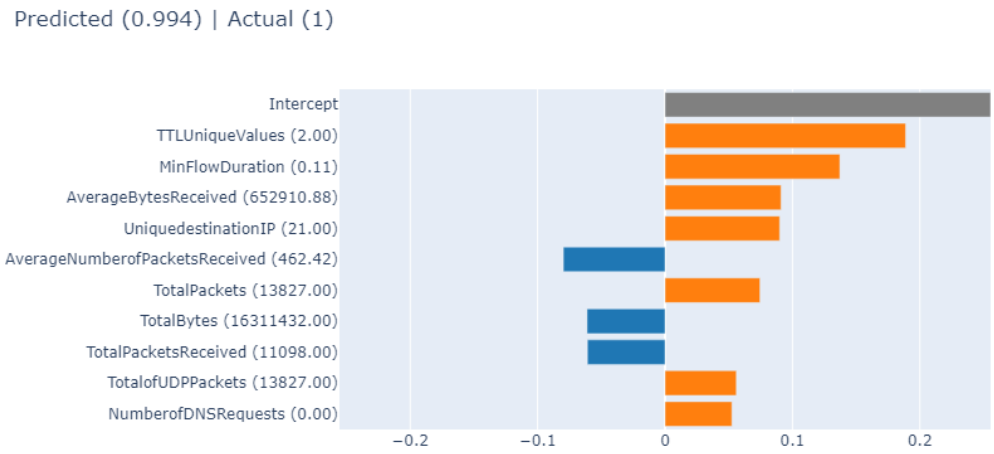


Figure 24. LIME Explanation for Data Sample Drawn from Wireless NAT Dataset

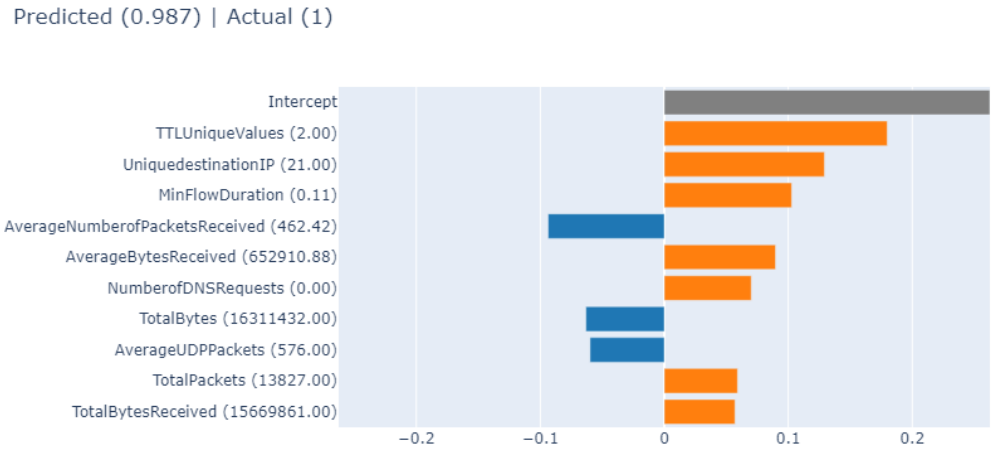


Figure 27. LIME Explanation for Data Sample Drawn from Obfuscated Version Wireless NAT Dataset

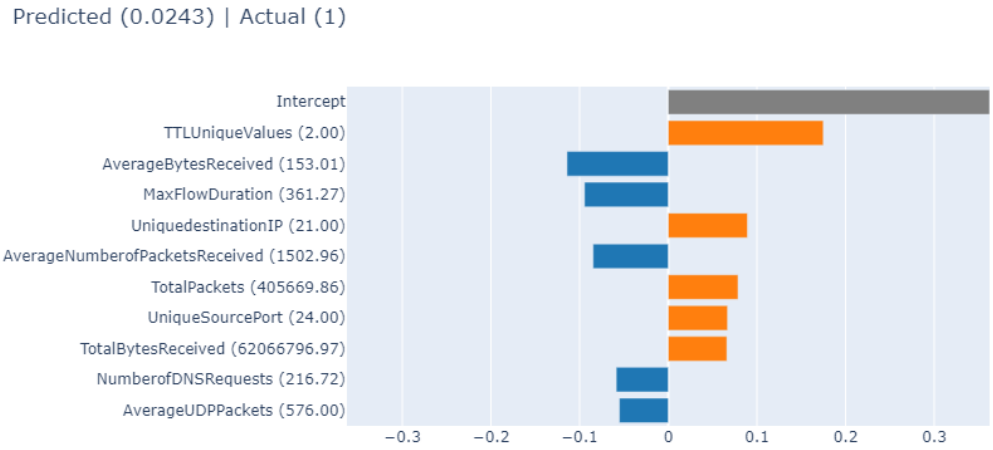


Figure 25. LIME Explanation when all Features are Obfuscated

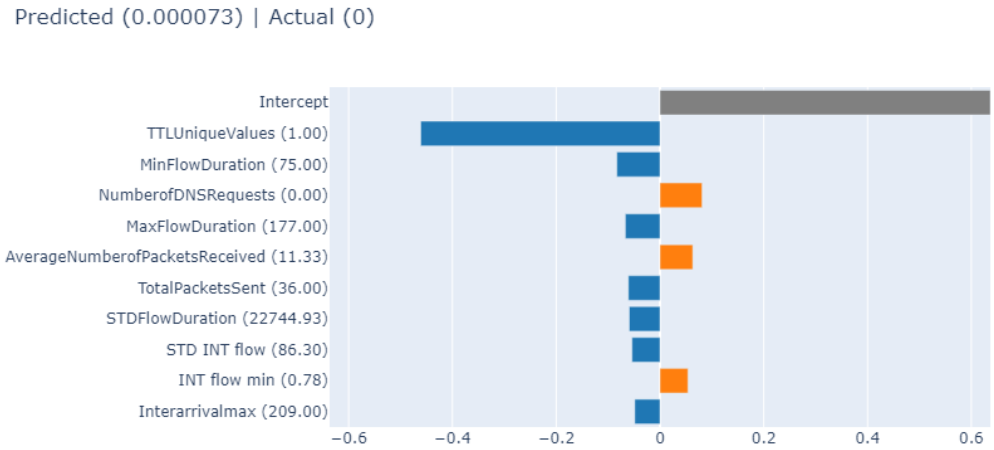


Figure 26. LIME Explanation for a Sample in Wired NAT Dataset

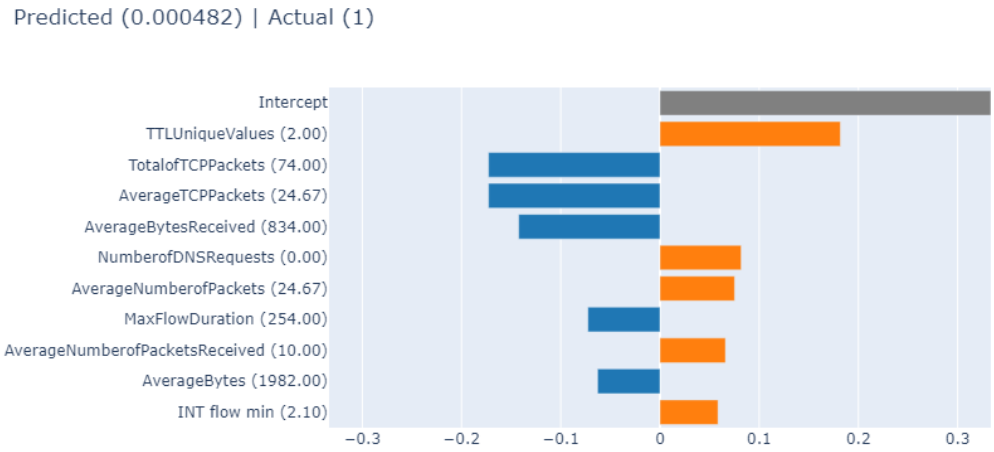


Figure 28. LIME Explanation for a Sample in Wired NAT Dataset

Figures 29 and 30 show two data samples taken from test two after removing all obfuscated features for the two classes for wireless NAT dataset. Both samples are predicted correctly with high efficiency. The LIME plot shows that a sample with one unique TTL value and small values of unique destination IP and port, and unique source port represent the class when there are <=2 hosts hidden. On the other hand, when the TTL unique values are greater than or equal to 2 and the unique destination IP and port, and unique source port values are high, the sample belongs to large hidden network. Figures 31 and 32 represent two data samples from wired NAT dataset, one predicted correctly and the other wrongly predicted. The one with wrong prediction has two TTL unique values and higher values of destination IP and source port than the decision rule but it corresponds to a small hidden network. The data taken from a new environment corresponds, in Figure 31, represents an ordinary host. The high number of destination IP and source port corresponds to a high activity. Whereas based on the model, which is trained on wireless NAT dataset, high activity means that there is a NAT device. Yet, this data sample is coming out from an ordinary host, and this leads to a false prediction.

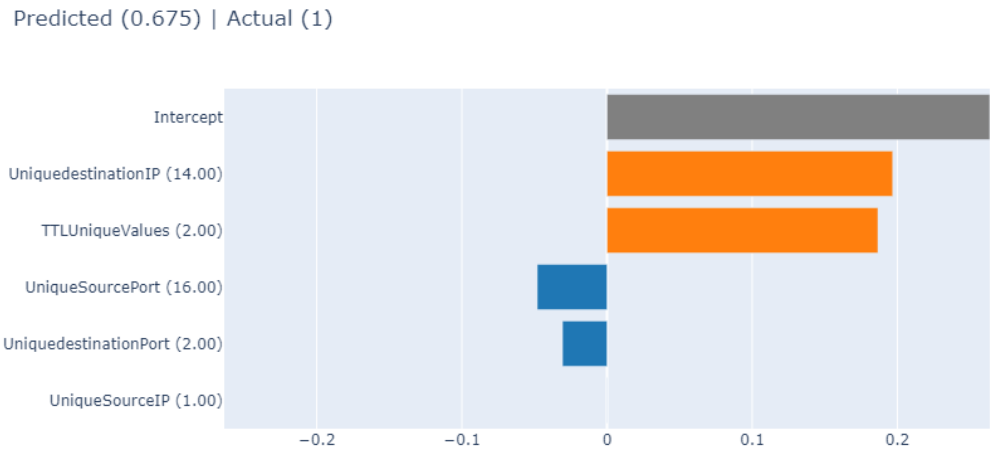


Figure 31. LIME Explanation for a Sample in Wireless NAT Dataset

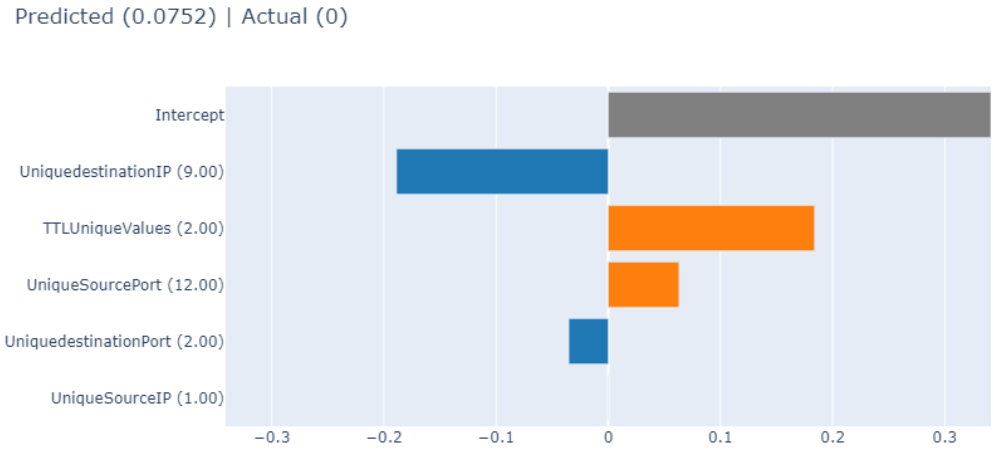


Figure 29. LIME Explanation for a Sample in Wireless NAT Dataset

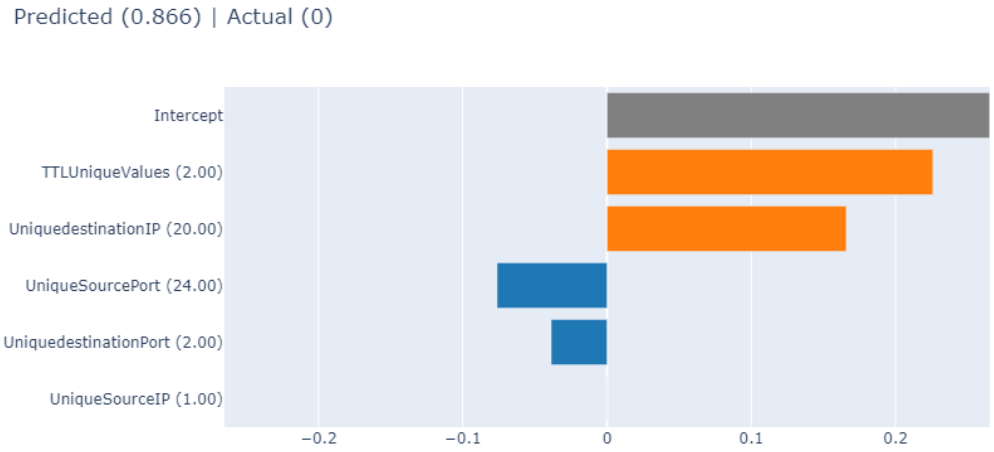


Figure 30. LIME Explanation for a Sample in Wired NAT Dataset

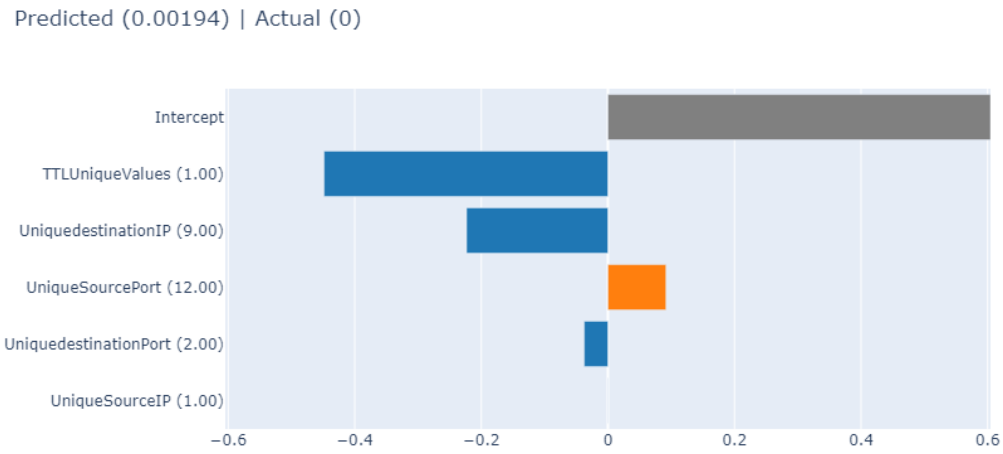


Figure 32. LIME Explanation for a Sample in Wired NAT Dataset

4.5 Comparison Between Benchmark and Proposed Method

Table 11 presents a comparison between some benchmark studies and our proposed approach. Previous studies either concentrate on NAT detection using ML or host counting. Each proposed approach yields excellent results on the datasets used by authors. Yet authors in [20] stated that their algorithm outperforms all previous methods. It is the first that focuses on detecting whether malicious traffic is coming from a single host or from multiple hosts. Authors in [20] stated that using the same data in training and testing the implemented algorithm performs much better than using another dataset in testing. This is also confirmed in our algorithm and the results are reported in tables 7 and 9 where the highest F1 score is when testing the model is evaluated using a test sample from the dataset used in training. Testing the model on a new data with different distributions leads to a reduction in model's performance. Although the model in [20] gives satisfactory results on their dataset, as illustrated before there is no guarantee that it will perform well on all network environments. In addition, the authors have stated that their method is not dependent on port numbers, but they have used the destination ports

as a feature to train the ML algorithm. Their method is limited to the number of classes used in training. Trying their algorithm in our dataset yields a 33.23% F1 score which is much less than the F1 score in both multiclass and binary host counting models. Authors in [21] implemented an algorithm to count and cluster IP traffic using machine learning. Unlike our proposed algorithm their algorithm works after separating traffic from different OSs. In addition, their algorithm works perfectly on their dataset, but it is incompatible with features obfuscation.

Table 11: Comparison Between our Proposed Algorithm and Previous Studies

Objective	Shakula et al [20]	Mateless et al [21]	Proposed algorithm
Detecting NAT devices	x	x	✓
Identifying network size or number of hosts hidden	✓	✓	✓
Handling traffic obfuscation	x	x	✓
Handling new environment	✓	x	✓
Handling malicious traffic	✓	x	x

4.6 Summary

This section presents the results and discussion of all implemented experiments in NAT detection and host number counting. It shows that the higher the time window aggregation the better the detection results. It shows that the implemented model cannot generalize and predict NAT or approximate devices in an unfamiliar environment. Yet after removing all aggregated features that could be changed due to obfuscation, the XGBoost network size detection model was able to generalize in the new environment. It is hard in the case of NAT detection to detect the exact number of devices. Host approximation is better than exact detection and it is not limited to a specific number of devices. Transfer learning has optimized the model such that is able to detect NAT in a

new network environment. When we have obfuscation, it is better to remove all obfuscated features because the obfuscation scenario used cannot be predicted.

CHAPTER 5

CONCLUSION

The main goal behind this thesis is to detect NAT devices through a supervised machine learning algorithm, approximate the number of hidden devices behind the detected NAT device using a machine learning approach, and study the generalizability of both models. This research was performed on a publicly available dataset [30] and the generalizability test was performed on obfuscated versions of the data and a new dataset collected at a wired network in the university labs. This thesis provides an ML approach to network features extracted from passively collected traffic data without being limited to a specific operating system. The model has high performance thus it can be used as a tool to help ISPs detect NAT devices and know how large the network is behind them.

The NAT detection model uses a machine learning algorithm to find features in the traffic data that give evidence for NAT presence. Among all machine learning algorithms implemented on different window sizes, the XGBoost classifier was the best at a time window equal to one minute. The detection model uses only features that influence NAT presence, this step is done through feature engineering and selecting the most appropriate features. The algorithm was performing much better than the traditional algorithm presented in [9]. The NAT detection algorithm is trained and tested on a dataset captured on different dates with a different number of hosts, different applications used, and different operating systems. This dataset is used to remove all limitations when assessing the model's ability to generalize. Based on the results it was shown that the model can detect NAT effectively on a dataset even after obfuscation, however, it fails to detect NAT devices in a new environment. After using explainable AI, it is extremely

hard to build a model that would be generalizable to general network environments since each environment has its characteristics. Even though the data set used to train the model has different variations, OS, and application uses, one cannot guarantee that another network will use the same operating systems or run the same application, or keep the application opened at the same time. Thus, packet sizes, number of packets, and interarrival time would be different. Besides after aggregation two datasets collected in different environments would not resemble each other. For this reason, transfer learning is used, and it is seen from the experiments that transfer learning builds an optimized model that has reliable performance in new environments.

The host counting model is also based on a machine learning approach that will search for patterns in traffic data that give evidence of the hidden network size. Two methods were adopted: exact host number identification and host count approximation. In host identification, XGboost and RF models show the best performance. The detection accuracy was low however, because the model was making wrong predictions for near classes. This is because we are depending on features presented in Table 2 and these features are highly affected by the number of packets sent in a flow, the size of packets, the duration, and when the application is opened. Thus, there is no guarantee that multiple users behind a NAT will be active at the same time. Thus, even though flow aggregation will decrease the number of false predictions, it was not able to prevent these predictions between near classes. So, the problem is transformed into a binary problem where the size of the network hidden is predicted. In this way, the model is not limited to a specific number of hosts. The XGBoost model gives a promising result in host approximation even after obfuscating the data. It also cannot generalize after removing obfuscated features. Yet when having all the features, it was hard to generalize because each

environment has its characteristic. In case of network size detection, it is better to remove all obfuscated features in case of obfuscation or environment change.

Using aggregated features improves the detection rate in both NAT detection and counting models. Both number of flows extracted per the time aggregation window size and the interarrival timing between flows are relevant to the detection algorithm. In both counting and NAT detection they are highly contributing based on the SHAP feature importance. These new features and the aggregation decrease the false predictions.

Based on this research there are some studies that should be tackled in the future:

1. Training and testing the counting model on data that contains more devices.
2. Find a way to remove the bias in exact host counting model and remove the limitation on the maximum number of hosts predicted.
3. Include IoT devices instead of using only desktops, laptops, and mobile phones.
4. Study if there is a way to normalize network traffic features such that the environment effect is removed.

REFERENCES

[1] S. KEMP, "Digital 2022: Global Overview Report," *Data Reportal*, *Kepios*, Jan. 26, 2022. [Online]. Available: https://datareportal.com/reports/digital-2022-global-overview-report. [Accessed: Aug. 17, 2022].

[2] P. Collela, "Ushering In A Better Connected Future," *BW Businessworld*, Jan. 16, 2017. https://www.businessworld.in/article/Ushering-In-A-Better-Connected-Future/16-01-2017-111504/. [Accessed: Aug. 17, 2022].

[3] S. Salomonsson, *Exploring NAT Host Counting Using Network Traffic Flow*, M.S. [Thesis], SE: Karlstads Univ., 2017. [Online]. Available: Digitala Vetenskapliga Arkivet.

[4] P. Srisuresh, M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations, RFC 2663," *RFC Editor*, *The Internet Society*, Aug. 1999. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2663. [Accessed: Jan. 20, 2021].

[5] H. Park, S. Shin, B. Roh and C. Lee, "Identification of Hosts behind a NAT Device Utilizing Multiple Fields of IP and TCP," in *Proc. of the 2016 Int. Conf. on Information and Communication Technology Convergence (ICTC), 2016, 19-21 Oct. 2016, Jeju, Korea (South)* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[6] S. M. Beellovin, "A Technique for Counting NATted Hosts," in *Internet Measurement Workshop: IMW '02: Proc of the 2nd ACM SIGCOMM Workshop on Inrernet Meaurement*, Nov. 6-8, 2002, Marseille, France, New York: Association for Computing Machinery, 2002, pp. 267-272.

[7] S. Abt, C. Dietz, H. Baier, and S. Petrovi, "Passive Remote Source NAT Detection Using Behavior Statistics Derived from NetFlow," in *Emerging Management Mechanisms for the Future Internet: AIMS 2013: Proc. of the 7th IFIP WG 6.6 Int. Conf. on Autonomous Infrastructure, Management, and Security, June 25-28, 2013, Barcelona, Spain*, G. Doyen, M. Waldburger, B. Stiller, P. Celeda, A. Sperotto, Eds. Berlin: Springer, 2013, pp. 148-159.

[8] L. Zhang, *Exploring NAT detection and host identification*, M.S. [Thesis], CA: Dalhousie Univ., 2018. [Online]. Available: Faculty of Graduate Sudies Online Theses.

[9] V. Krmicek, Jan Vykopal, and R. Krejci, "NetFlow Based System for NAT Detection," in *Co-Next Student Workshop: Proc. of the 5th Int. Student Workshop on Emerging Networking Experiments and Technologies*, Dec. 1, 2009, Rome, Italy, New York: Association for Computing Machinery, 2009, pp. 23-24.

[10] P. Phaal, "Detecting NAT Devices using sFlow," p. 4, 2003. [Online]. Available: https://sflow.org/detectNAT/.

[11] L. Orevi, A. Herzberg, and H. Zlatokrilov, "DNS-DNS: DNS-Base De-NAT Scheme," in *Cryptology and Network Security: CANS 2018: Proc. of the 17th Int. Conf. on Cryptology and Network Security, Sep. 30 – Oct. 3, 2018, Naples Italy*, J. Camenisch, P. Papadimitratos, Eds. Berlin: Springer, 2018, pp. 69-88.

[12] J. Bi, L. Zhao, and M. Zhang, "Application Presence Fingerprinting for NAT-Aware Router," in *Knowledge-Based Intelligent Information and Engineering*

Systems: KES 2006: Proc. of the 10th Int. Conf. on Knoweledge-Based and Intelligent Information and Engineering Systems, Oct. 9-11, 2006, Bournemoth, UK, B. Gabrys, R. J. Howlett, L. C. Jain, Eds. Berlin: Springer, 2006, pp. 678-685.

[13] B. Yan, L. Huang, G. Gou, Y. Guo, and Y. Bao, "A Fine-Grained Large-Scale NAT Detection Method," in *Advanced Multimedia and Ubiquitous Enfineering, Lecture Notes in Electrical Engineering*, Springer, vol. 339, pp. 493-499, 2016.

[14] A. Luti, M. Bagnulo, A. Dhamdhere, and K. C. Claffy, "NAT Revelio: Detecting NAT444 in the ISP," in *Passive and Active Measurement: PAM 2016: Proc. of the 17th Int. Conf. on Passive and Active Network Measurement, March 31 – April 1, 2016, Heraklion, Greece*, T. Karagiannis, X. Dimitropoulos, Eds. Switzerland, Springer, 2016, pp. 149-161.

[15] T. Komárek, M. Grill and T. Pevný, "Passive NAT detection using HTTP access logs," in *Proc. of the 2016 IEEE Int. Workshop on Information Forensics and Security (WIFS), Dec. 4-7, 2016, Dhabi, United Arab Emirates* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[16] Z. Yan, N. Yu, H. Wen, Z. Li, H. Zhu, and L. Sun, "Detecting Internet-Scale NATs for IoT Devices Based on Tri-Net," in *Wireless Algorithms, Systems, and Applications: WASA 2020: Proc. of 15th Int. Conf. on Wireless Algorithms , Systems, and Applications, Sep 13-15, 2020, Qingdao, China*, D. Yu, F. Dressler, J. Yu, Eds. Switerland, 2020, pp. 602-614.

[17] A. Safari Khatouni, L. Zhang, K. Aziz, I, Zincir, and N. Zincir-Heywood, "Exploring NAT Detection and Host Identification Using Machine Learning," in *Proc. of the 2019 15th Int. Conf. on Network and Service Management (CNSM), Oct. 21-25, 2019, Halifax NS, Canada* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[18] S. Lee, S. Kim, J. Lee, B Roh, Byeong-hee, "Supervised Learning-Based Fast, Stealthy, and Active NAT Device Identification Using Port Response Patterns," *Symmetry*, vol. 12, no. 9, Sep. 2020.

[19] Li Rui, Zhu Hongliang, Xin Yang, Luo Shoushan, Yang Yixian, and Wang Cong, "Passive NATted Hosts Detect Algorithm Based on Directed Acyclic Graph Support" in *Proc. of the 2009 Int. Conf. on Multimedia Information Networking and Security, Nov. 18-20, 2009, Wuhan, China* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[20] S. Shukla and H. Gupta, "Identification and Counting of Hosts Behind NAT Using Machine Learning," *SN Computer Science*, vol. 3, Jan. 2022.

[21] R. Mateless, H. Zlatokrilov, L. Orevi, M. Segal, and R. Moskovitch, "IPvest: Clustering the IP Traffic of Network Entities Hidden Behind a Single IP Address Using Machine Learning," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3647-3661, Sep. 2021.

[22] Y. Gokcen, V. A. Foroushani, and A. N. Z. Heywood, "Can We Identify NAT Behavior by Analyzing Traffic Flows?," in *Proc. of the 2004 IEEE Security and Privacy Workshops, May 17-18, 2014, Jose, CA, USA* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[23] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, taxonomies,

opportunities and and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 85-115, June 2020.

[24] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable Artificial Intelligence (XAI) to Enhance Trust Management in Intrusion Detection Systems Using Decision Tree Model," *Complexity*, vol. 2021, Article ID 6634811, Jan. 2021.

[25] M. Sarhan, S. Layeghy, and M. Portmann, "An Explainable Machine Learning-based Network Intrusion Detection System for Enabling Generalisability in Securing IoT Networks," *arXiv preprint*, arXiv:2104.07183v1. April 2021.

[26] B. Kulis, K. Saenko, T. Darrell, "What You Saw is ot What You Get: Domain Adaptation Using Asymmetric Kernel Transforms," in *Proc. of the CVPR 2011, June 20-25, 2011, Colorado Springs, CO, USA* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[27] B. Long, Y. Chang, A. Dong, J. He, "Pairwise Cross-Domain Factor Model for Heterogeneous Transfer Ranking", *WSDM '12*, Feb. 8, 2012.

[28] J. Zhao, S. Shetty, J. W. Pan, C. Kamhoua, and K. Kwiat, "Transfer Learning for Detecting Unknown Network Attacks," *EURASIP Journal on Information Security*. 2019.

[29] Juan Zhao, Sachin Shetty, and Jan Wei Pan, "Feature-Based Transfer Learning for Network Security," in *Proc of MILCOM 2017 – 2017 IEEE Military Communications Conference (MILCOM), Oct. 23-25, 2017, Baltmoree, MD, USA* [Online]. Available: IEEE Xplore, https://ieeexplore.ieee.org.

[30] S. Farhat, I. H. Elhajj, and A. Kayssi, "NAT NETWORK TRAFFIC DATASET," *IEEE DataPort*, Sep. 2020. [Dataset]. Available: https://ieee-dataport.org/documents/nat-network-traffic-dataset. [Accessed 11 December 2020].

[31] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5-31, Oct 2001.

[32] D. R. Cutler, T. C. Edwards, K. H. Beard, A. Cutler, K. T. Hess, J. Gibson, and J. J. Lawler, "Random Forests for Classification In Ecology," *Ecology*, vol. 88, no. 11, pp. 2783-2792, Nov. 2007.

[33] P. D. Caie, N. Dimitriou, and O. Arandjelović, "Precision Medicine in Digital Pathology Via Image Analysis and Machine Learning," in *Artificial Intelligence and Deep Learning in Pathology*, S. Cohen, Elsevier, 2021, pp. 149-173.

[34] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends," *Neurocomputing*, vol. 401, pp. 189-215, Sep. 2020.

[35] "Advances in Computers," Advances in Computers | The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases | ScienceDirect.com by Elsevier. [Online]. Available: https://www.sciencedirect.com/bookseries/advances-in-computers/vol/117/issue/1. [Accessed: 13-Jul-2021].

[36] S. Misra and H. Li, "Noninvasive Fracture Characterization Based on the Classification of Sonic Wave Travel Times," Machine Learning for Subsurface Characterization, pp. 243–287, 2020. [Online] Available: Google Books.

[37] O. Harrison, "Machine Learning Basics with the K-Nearest Neighbors Algorithm," Medium, 14-Jul-2019. [Online]. Available:

https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. [Accessed: 13-Jul-2021].

[38] “How XGBoost Works,” AWS. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html. [Accessed: 15-Nov-2022].

[39] “XGBoost for Classification,” Vertica. [Online]. Available: https://www.vertica.com/docs/10.1.x/HTML/Content/Authoring/AnalyzingData/MachineLearning/XGBoost/XGBoostForClassification.htm. [Accessed: 15-Nov-2022].

[40] T. Shin, “How to Explain Each Machine Learning Model at an Interview,” Medium, 21-Jun-2020. [Online]. Available: https://towardsdatascience.com/how-to-explain-each-machine-learning-model-at-an-interview-499d82f91470. [Accessed: 15-Nov-2022].

[41] S. Dash, O. Günlük, and D. Wei, “Boolean Decision Rules via Column Generation,” arXiv.org, 05-Aug-2020. [Online]. Available: https://arxiv.org/abs/1805.09901. [Accessed: 13-Jul-2021].

[42] H. Nori, S. Jenkins, P. Koch, and R. Caruana, “InterpretML: A Unified Framework for Machine Learning Interpretability,” arXiv.org, 19-Sep-2019. [Online]. Available: https://arxiv.org/abs/1909.09223. [Accessed: 10-Jul-2020].

[43] N. Segev, M. Harel, S. Mannor, K. Crammer, and R. El-Yaniv, " Learn on Source, Refine on Target: A Model Transfer Learning Framework with Random Forests," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 9, pp. 1811-1824, 2016.