Exploring NAT Detection and Host Identification Using Machine Learning

Ali Safari Khatouni[1], Lan Zhang[2], Khurram Aziz[3], Ibrahim Zincir[4], Nur Zincir-Heywood[5]
[1] Dalhousie University, Canada ali.safari@dal.ca, [2] Dalhousie University, Canada ln527401@dal.ca,
[3] Dalhousie University, Canada kaziz@dal.ca, [4] Yasar University, Turkey ibrahim.zincir@yasar.edu.tr,
[5] Dalhousie University, Canada zincir@cs.dal.ca

***Abstract*—The usage of Network Address Translation (NAT) devices is common among end users, organizations, and Internet Service Providers. NAT provides anonymity for users within an organization by replacing their internal IP addresses with a single external wide area network address. While such anonymity provides an added measure of security for legitimate users, it can also be taken advantage of by malicious users hiding behind NAT devices. Thus, identifying NAT devices and hosts behind them is essential to detect malicious behaviors in traffic and application usage. In this paper, we propose a machine learning based solution to detect hosts behind NAT devices by using flow level statistics (excluding IP addresses, port numbers, and application layer information) from passive traffic measurements. We capture a large dataset and perform an extensive evaluation of our proposed approach with four existing approaches from the literature. Our results show that the proposed approach could identify NAT behaviors and hosts not only with higher accuracy but also demonstrates the impact of parameter sensitivity of the proposed approach.**

I. Introduction

NAT devices are commonly used by Internet Service Providers (ISPs), enterprises and end users in their Local Area Networks (LAN) for a group of hosts. However, they can also be used just for one host. In home networks, most ISPs provide WiFi-enabled NAT home gateways to their users. Thus, when users connect their devices to the Internet, their private Internet Protocol (IP) addresses are hidden since they are replaced with a public IP address that the NAT provides. In addition to anonymity, NAT also provides a lower cost alternative to the rising prices of public IP addresses that have now reached 20$/year per IP[1]. This is especially attractive for ISPs who are more and more looking into NAT solutions to mask part of their network. Carrier-Grade NAT (CGN) is one such technology that allows ISPs to limit the use of expensive public IP addresses. For instance, Safari et al. [1] show that many of the mobile operators in Europe use CGN in their operational networks while mobile users get private IP addresses. NATs are, therefore, gaining popularity for many reasons such as a solution to the shortage of IPv4 addresses; or for security and privacy reasons as they provide anonymity within a group of computers behind a NAT device. However, the same reasons also make it attractive for malicious users who want to hide their real identities. Hence, NAT usage increases both in legitimate and in malicious environments. Rajab et al. [2] show that the widespread use of NAT had significant implications on how different families of malware spread on the Internet. Maier et al. [3] show that 20 percent of the digital subscriber lines may contain multiple hosts behind one address.

In this research, our goal is to find specific patterns in the network traffic that enables us to identify NAT like behaviors. To this end, we propose a machine learning (ML) based approach to automatically find patterns indicating NAT usage without using IP addresses, port numbers, and application layer information, i.e. excluding application layer headers and payload. In doing so, our aim is to not only analyze how far we can push a well-generalized ML approach but also to benchmark the proposed solution against four other approaches from the literature, under encrypted payload and traffic filtering conditions. To the best of our knowledge, this is the first time generalization of ML-based detectors are benchmarked under encrypted payload and traffic filtering conditions for NAT behavior detection. This approach is inspired by the work by Gokcen et al. [4]. They extract flow based features using a specific tool, namely Netmate. We believe that there are several more recent and advanced network flow analyzers that can potentially improve the detection accuracy. To this end, we evaluate ten different ML classifiers using four different feature sets extracted from well-known network flow analyzers such as Argus, Tranalyzer, Tstat, and Netmate. Moreover, we compare these results to four other approaches proposed in the literature in [4], [3], and [5]. Results show that the proposed solution outperforms the existing approaches by achieving a 96% precision with 97% recall. Further analysis also sheds light into the parameter sensitivity of the proposed solution in terms of its accuracy.

The remainder of this paper is organized as follows. Section II reviews the literature on NAT detection systems. Section III describes the passive traffic measurement setup and the generated dataset. Section IV details the proposed approach while evaluations are reported and the results are presented in Section V. Finally, conclusions are drawn and future work is discussed in Section VI.

II. Related Work

The identification of NAT devices and the number of end users behind such devices is an interesting research area. On one hand, NAT technology is employed to make better use

[1]http://www.ipaddressnews.com/2019/03/30/629

978-3-903176-24-9

of the available IPv4 address space of an organization. On the other hand, it is employed by users (with and without malicious intent) to hide their identity and anonymous access to the Internet. Thus, different approaches are proposed, but generally, researchers used some form of application header information to identify different NAT behaviors. In the following, we summarize the related work in terms of (i) Detecting NAT behaviors, and (ii) Detecting hosts behind the NAT devices.

A. Detecting NAT behaviours

Current NAT detection research is mainly based on NAT behavior modeling. To this end, different machine learning algorithms are adopted to build NAT behavior models from end hosts. Gokcen et al. focus on exploring specific patterns in the network traffic that can identify NAT like behaviors [4]. They use NetMate[2] to generate flows and ML approaches to find patterns indicating NAT usage. They exclude IPID[3] [6], port numbers and payload information.

Komárek et al. focus on detecting NAT behaviors based on IP-based features in HTTP access logs [5]. These include the number of unique contacted IP addresses, number of unique user-agents, number of unique operating systems and versions, number of unique browsers and versions, number of persistent connections, number of upload bytes, number of download bytes, and number of sent HTTP requests. Andra et al. present an active measurement approach to detect CGN deployed at the ISPs, without any prior knowledge of the environment [7]. Muller et al. discover cascaded NATs on the path between two arbitrary peers on the Internet using UDP packets [8]. Krmicek et al. [9] and Abt et al. [10] use passive measurement approaches to detect NAT behaviors using Netflow based traffic features such as Time to Live (TTL), IP Address, and TCP SYN packet size.

B. Detecting Hosts behind the NAT devices

Paxton et al. identify network packets across network address translational boundaries to identify hosts [11]. They rely on the fact that translational boundaries, i.e., NAT devices, work by altering packet headers instead of packet payloads. This fact allows matching packets across NAT devices. However, this only applies when the traffic is captured on both sides of a NAT device. Thus, it is unable to identify end hosts behind a NAT device from one vantage point. Mongkolluksamee et al. count the number of the end hosts behind a NAT device [12], which implements a per-flow IPID sequence; a random IPID; or a global IPID based on the sequence of IPID, a TCP sequence number, and a TCP source port in different manners. Considering operating systems use different approaches to assign IPID, TCP sequence number, and TCP source port number, they can only count the number of hosts but cannot identify them. Bursztein et al. [13] present a similar approach to counting the number of hosts behind the NAT using the TCP timestamp (TSval) options [14]. This is a technique of IP agnostic real-time traffic filtering and host identification using TCP timestamps. It uses a function on the TCP timestamp value to identify hosts [15]. Castiglione et al. [16] focus on device tracking in private networks via Network Address and Port Translation (NAPT) log analysis. They determine a host profile or fingerprint. They track down the movement of the device and aim to identify the specific host behind one private network without its IP address. Verde et al. [17] build a fingerprinting framework to identify users behind a NAT device using NetFlow records. Weaver [18] models the scanning behavior of botnets by counting the number of devices behind an IP address space. Maier et al. [3] propose a unique approach for detecting the presence of NAT and for estimating the number of hosts connected behind a NAT gateway using IP TTLs and HTTP user-agent strings.

[2]https://github.com/DanielArndt/netmate-flowcalc

[3]IPID is a 16-bit counter that identifies unique packets when fragmentation occurs.

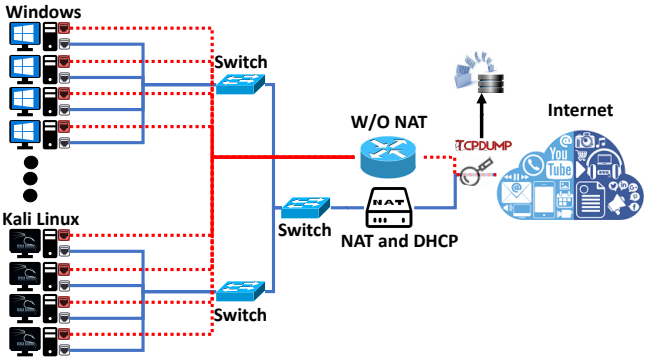


Fig. 1: The passive measurement network setup.

On the other hand, in this paper, we follow a passive measurement based approach for identifying NAT behaviours. In doing so, we aim to minimize the overhead on the network. This work differs from previous works in that the proposed system does not employ any application layer information such as HTTP headers, IP addresses, port numbers, or TTL information. Instead, the proposed system employs only traffic flow features extracted from the traffic flow analyzers (excluding the features given above), Furthermore, comparisons are performed against four other existing systems in the literature on publicly available large datasets to explore the generalization of the proposed approach compared to the state-of-the-art in the field.

III. Passive Traffic Measurements

In order to explore and understand different NAT and host behaviors, we set up an experimental testbed to collect a large dataset. This then enabled us to perform an extensive evaluation of different NAT identification approaches from the literature against the proposed approach.

A. Experiment setup

Figure 1 shows the overall view of the testbed - passive measurement network - configuration. The leftmost side shows

the different hosts (end systems), where each of them is connected to two network interfaces. There are 16 hosts equipped with either Kali Linux Rolling[4] or Microsoft Windows 7 Professional operating systems. Furthermore, there are two network configurations for the testbed set up: (i) Network with NAT; and (ii) Network without NAT. In the first configuration, each host is connected to a switch then these connections pass through a NAT and Dynamic Host Configuration Protocol (DHCP) server to access the public Internet. This is presented by the solid blue lines. In the second configuration, all hosts are connected to a router without going through the NAT and the traffic is forwarded to the public Internet (dotted red lines). The traffic is captured after the NAT device by means of *tcpdump*. In doing so, the aim is to generate and collect data to represent real-life scenarios for exploring and understanding different NAT and host behaviors. This also provides us a dataset for simulating forensic analysis type of scenarios. In this case, an analyst could need to analyze a traffic dataset to identify any host that might have anonymized its identity by using a NAT system. This is a challenging scenario for NAT detection systems because the operating systems, web browsers, etc. of the hosts could be ambiguous attributes to be used in order to detect the NAT type behaviors.

To be able to generate realistic traffic, we employed iMacros[5] which is an extension for web browsers (both Google-Chrome and Firefox) for automatic URL generation and file download. We use iMacros and customized JavaScript to automatically search random keywords on Google. Once the search results are received, the script randomly selects one of the URLs to visit, then follows another URL from that web page, and repeats the process five more times. Then, the script closes the current webpage it has visited and opens a new one to start the process all over again by searching for a random keyword on Google. The program repeats the above steps, and when it encounters files that can be downloaded, it downloads them. The script is run for about 10 hours for each batch of the experiment. This experiment is run for 17 days to collect the data employed in this work.

B. Dataset

In this research, we collect a large dataset that contains multiple packet traces, i.e. a `pcap` file for each day of the experiment that we run. The total size of the raw `pcap` files is 300 GB. As discussed earlier, during these scenarios, we captured packet traces for two scenarios: (i) With NAT; and (ii) Without NAT. This enabled us to generate and collect a big dataset with ground truth information in terms of different behaviors including NAT hosts.

IV. Methodology

In this section, we present our methodology and the performance metrics used.

The first step of all ML-based approaches is to preprocess the dataset and create a fairly large and representative ground truth dataset. The dataset collection and its characteristics are described in Section III. We have used this dataset to evalutae and bechmark different approaches againts the proposed approach. These include approaches that use application layer information: (i) different browsers versions; (ii) different browsers; (iii) different operating systems; and that do not use application layer information where (iv) traffic flow information is used instead. Table I shows the approaches that are evaluated to identify NAT behaviours. Approach A1 aggregates the number of different browser versions and the operating system information for each IP address in a given time window. If it detects more than one browser version then it classifies that host as a device behind the NAT. On the other hand, A2 aggregates the number of different browsers and the operating system whereas A3 aggregates only the number of different operating systems. A4 is based on [5], where eight features are extracted for each host identified in the traffic trace. These are then input to a decision tree classifier to identify hosts behind the NAT. These features include the number of unique contacted IP addresses, the number of unique user-agents, the number of unique Operating Systems and their versions, the number of unique browsers and their versions, the number of persistent connections, the number of upload bytes, the number of download bytes, and the number of HTTP requests sent.

TABLE I: Overview of the approaches employed.

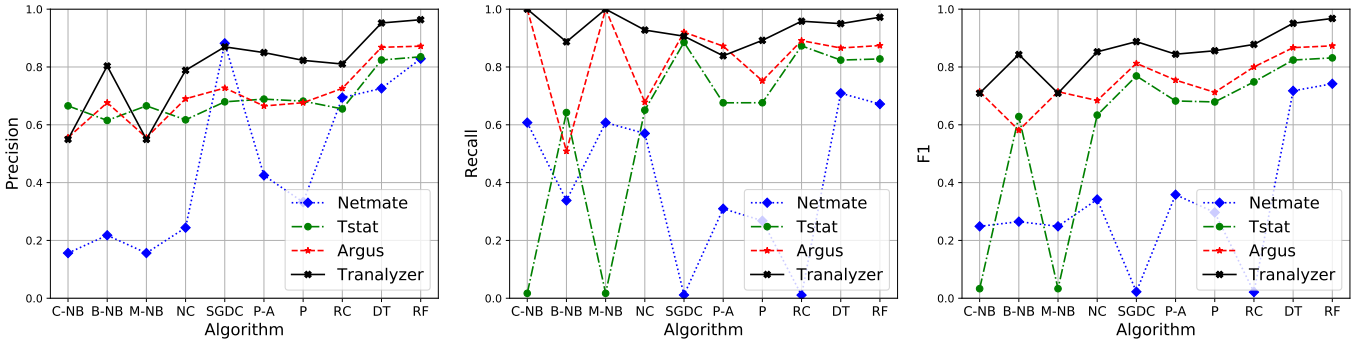
Approach	IP header	HTTP header
A1	IP address, TTL	OS, Browser, Browser Version
A2	IP address, TTL	OS, Browser
A3	IP address, TTL	OS
A4	IP	OS, Browser, Browser Version, Request, Keep-Alive
A5	-	Several number of flow level statistics

Last but not the least, in our proposed approach, A5, we opted to extract flow level information using four well-known network traffic analyzers, i.e., Argus [19], Netmate [4], Tranalyzer [20], and Tstat [21]. Gokcen et al. [4] showed that the flow level statistics could obtain higher accuracy with respect to other approaches that used operating system and application level information. Moreover, Khatouni et al. [22] showed that different feature sets obtained by flow analyzers have a significant impact on the encrypted traffic classification. Traffic flow exporters, namely Argus, Netmate, Tranalyzer, and Tstat extract different number of features - 51, 40, 85, and 110 features from each TCP/IP flow, respectively.

In this work, we explore whether the conclusions drawn in Khatouni et al. [22] are also valid in NAT detection or not. Therefore, we compare the performance of four different traffic flow feature sets obtained using different flow analyzers. Next, we apply 10 different ML classifiers on these feature sets and run extensive evaluations. Table II presents the different ML classifiers and their abbreviations used in the paper. In order to employ these techniques, the dataset is divided into two disjoint sets as training (70%) and testing (30%), respectively.

[4]https://www.kali.org/news/kali-linux-2017-2-release/

[5]https://imacros.net



(a) Precision of ML classifiers. (b) Recall of ML classifiers. (c) F1 Score of ML classifiers.

Fig. 2: Accuracy of different ML classifiers using different traffic flow feature sets.

TABLE II: The list of ML classifiers employed.

#	ML Classifier	Abbreviation
1	Complement Naive Bayes	C-NB
2	Naive Bayes	B-NB
3	Multinomial Naive Bayes	M-NB
4	Nearest Centroid	NC
5	linear Models with Stochastic Gradient Descent	SGDC
6	Passive Aggressive	P-A
7	Perceptron	P
8	Classifier using Ridge Regression	RC
9	Decision Tree	DT
10	Random Forest	RF

A. Performance metrics

We use the confusion matrix to present the accuracy of the classifier. Confusion matrix provides the number of flows classified as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Furthermore, we calculate the Precision, Recall and F1 Score based on the confusion matrix and generated the Receiver Operating Characteristic (ROC) curves.

- Precision: Defines the ability of the classifier not to label a negative sample as positive, Eq. 1.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (1)$$

- Recall: Defines the ability of the classifier to find all the positive samples, Eq. 2.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (2)$$

- F1 Score: Defines the harmonic mean of the precision and recall, Eq. 3.

$$F1\ Score = 2 * \frac{(Precision * Recall)}{Precision + Recall} \quad (3)$$

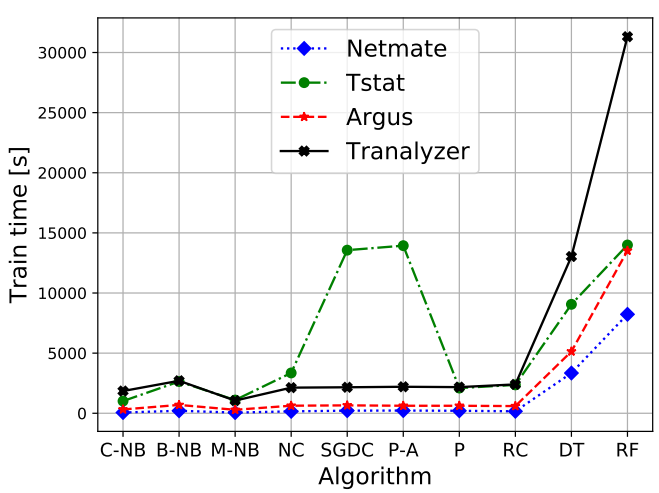
- ROC curve: The ROC curve is generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings. The TPR is also called as recall or sensitivity in ML. The FPR is also known as the probability of false alarms. The ROC curve is the sensitivity as a function of the probability of false alarms. Mainly, if the probability distributions for both the detection and the false alarm are known, then the ROC curve plots the Cumulative Distribution Function (CDF) of the detection probability in the y-axis versus the CDF of the false-alarm probability on the x-axis. The best decision rule is high on sensitivity and low on the probability of false alarms.

V. Results

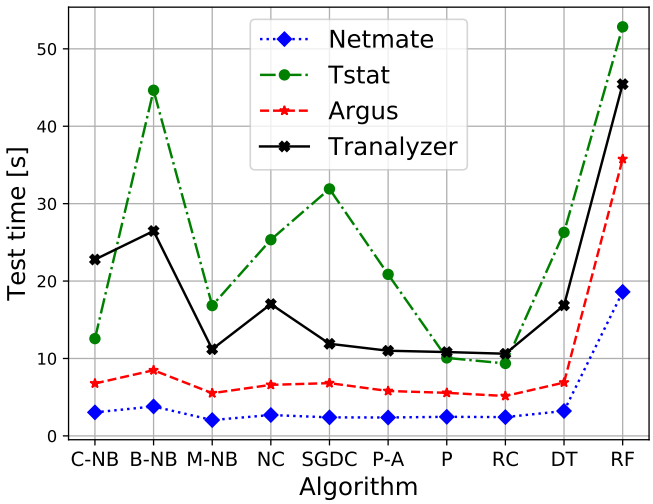
In this section, we present the results of the extensive evaluations of the proposed solution.

A. Model selection

We first explore the impact of the ML classifiers. Figure 2 shows the evaluation of the 10 ML classifiers using three performance metrics; Precision, Recall, and F1 score in figures 2a, 2b, and 2c, respectively. Fig. 2c presents the F1 score on the test dataset (y-axis) for different ML classifiers (x-axis). It shows the results obtained by using four feature sets, namely Argus, Netmate, Tranalyzer, and Tstat. Independent from the traffic flow exporter tool used, RF classifier provides the highest value and the DT classifier gets the second highest. The solid black line reports the result from the Tranalyzer feature set and it outperforms all the others. Generally, training time is an important factor/parameter while choosing an ML classifier for a given task. To this end, Figure 3 shows the training and test time (y-axis) for different ML classifiers (x-axis). Both the training and test time of the RF classifier is two times more than the DT classifier with less than 3% improvement in accuracy (see Figure 2a). According to these results, Tranalyzer feature set seems to provide the highest accuracy in terms of Precision, Recall and F1 Score no matter which ML algorithm is used. Moreover, among the ML algorithms, DT classifier provides the second-best accuracy in terms of Precision, Recall and F1 Score - no matter which features set is used. Given that the DT classifier seems to achieve good performance with much less training and testing time - hence minimum delay - compared to RF classifier in our experiments, we chose the DT classifier as the best performing one in terms of Precision, Recall and F1 Score in this work. Thus going forward, we just employ the DT classifier with the Tranalyzer feature set, since this combination achieved the best



(a) Training time using different flow feature extraction tools.



(b) Testing time using different flow feature extraction tools.

Fig. 3: Training and testing time of different ML classifiers using different feature sets.

performance with minimum delay (fastest time) compared to the other traffic flow feature sets used.

We then analyzed the effect (if any) of the application layer information using HTTP header as well as TTL on the NAT detection. To this end, we compare the application layer based approaches given in Table I. Figure 4 shows the precision (right plot), the recall (center plot), and the F1 Score (left plot), for values of $Window \in [0, 100]$ for the approaches using the window, i.e. A1, A2 and A3. The window equals to 0 means that we consider window size equal to the whole dataset. These results show that the window size seems to have a slight impact on accuracy. Intuitively, the window size of 1 second shows the lowest accuracy because the probability of having at least two active users at the same second is lower than a longer time window. Among these three approaches, A1 obtains the highest accuracy. On the other hand, A4, which uses web browser related information more than the other approaches (see Table I), obtains Precision, Recall, and F1 Score equal to 0.96, 0.76, and 0.84, respectively. Thus, it has the best performance among the approaches using the application layer and TTL information. However, its performance is still lower than the performance of the DT classifier using the Tranalyzer feature set, Table III.

Fig. 5 illustrates the top 10 features selected by the DT classifier using the Tranalyzer feature set. This presents the top 10 important features as the normalized average impurity computed by the information gain algorithm used by the DT classifier. The x-axis of Fig. 5 is detailed further in Table IV, where the first column of the table corresponds to the x-axis of Fig. 5. Table IV presents the description of the top 10 features extracted by the Tranalyzer, traffic flow exporter. This shows that the classifier could identify the hosts behind the NAT based on the packet sizes, the number of concurrent connections, the amount of data uploaded and downloaded as well as the number of packets and the inter-arrival time between packets.

TABLE III: The summary of the performances of the five approaches employed.

Approach	Precision	Recall	F1 Score
A1	0.39	0.55	0.46
A2	0.16	0.16	0.16
A3	0	0	0
A4	0.96	0.76	0.84
A5 (Tranalyzer)	0.96	0.97	0.96

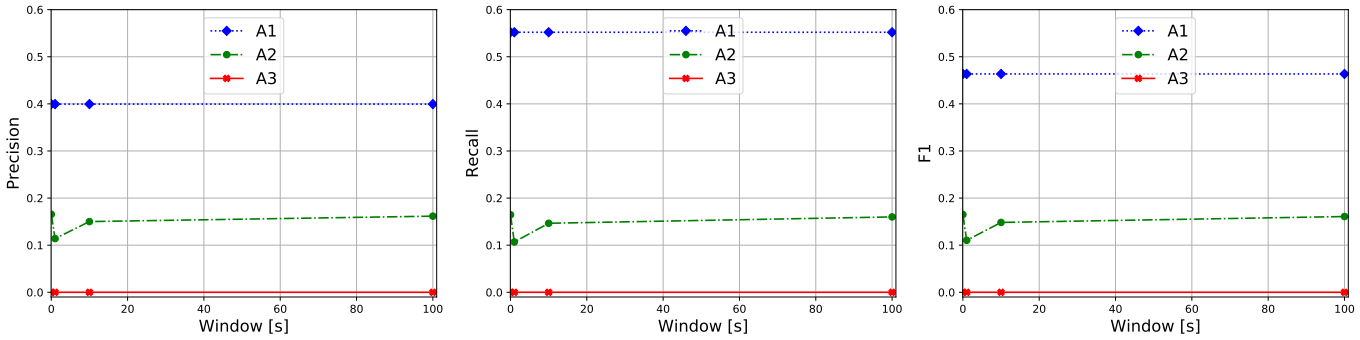
Furthermore, the training dataset size could be an important factor/parameter for the ML classifier in order to understand any possibility of overfitting, etc. Fig. 6 shows the learning curve of the DT classifier. The y-axis represents the overall accuracy score and the x-axis represents the size of the training dataset in millions [M]. These results show a slight improvement in the test accuracy from 93% to 95% when the training dataset size increases from 0.25 M to 2 M.

Table III shows a summary of the best performance obtained by each approach evaluated. It should be noted here that our proposed approach, A5, achieves the highest accuracy without using IP addresses, port numbers, or any operating system information and application layer headers.

B. *Parameter tuning*

In this section, we investigate important parameters of the DT classifier to obtain the optimal parameter set. Python [6] implementation of the DT classifier supports two criteria to measure the quality of a split: the Gini Impurity measure and the Information Gain measure [23]. The Gini Impurity measure is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled

[6]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html



(a) Precision (b) Recall (c) F1 Score

Fig. 4: The accuracy of three approaches using different types of application layer information.

x-axis	Tranalyzer
1	$S->C$ Number of connections from destination IP to different hosts
2	$S->C$ TCP maximum packet size
3	$C->S$ the f number, experimental: connSipDprt/connSip
4	$S->C$ TCP minimum effective window size
5	$S->C$ Number of connections from a host IP to different destinations
6	$C->S$ TCP maximum effective window size
7	$C->S$ Number of connections source IP and destination IP
8	$C->S$ Number of connections source IP and destination port
9	$S->C$ TCP maximum effective window size
10	$S->C$ Number of connections source IP and destination port

TABLE IV: List of top ten most important Trnalyzer features as selected by the DT classifier. $S->C$ denotes the Server to Client direction and $C->S$ denotes the Client to Server direction.

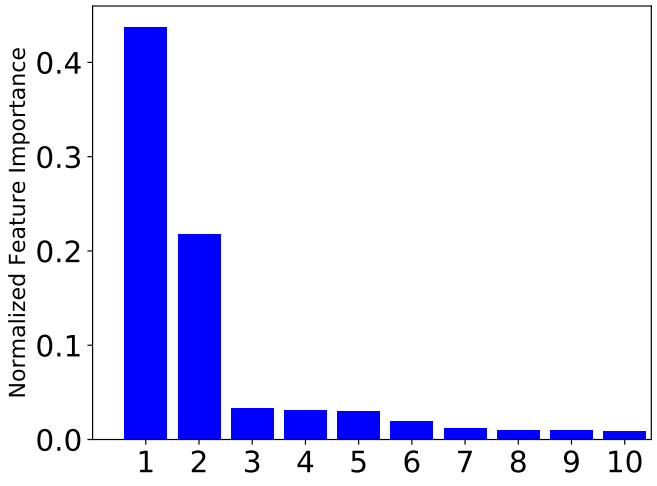


Fig. 5: The top 10 most important features using DT to calculate feature importance on Tranalyzer feature set.

according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability of a mistake in categorizing that item. It obtains its minimum value when all cases in the node fall into a single target category.

On the other hand, the Information Gain measure is based on the concept of entropy and information content from information theory. Information gain is used to decide on which feature to split at each step in building the decision tree. The information gain measure of a certain event is the discrepancy of the amount of information before someone observes that event and the amount after observation.

Fig. 7 shows the ROC curve for the DT classifier with the Gini Impurity and the Information Gain split criteria. The y-axis and x-axis illustrate the TPR and FPR, respectively. As depicted in Fig. 7, these two criteria have a negligible impact on the ROC curve, the area under the curve shows 0.001 difference.

The tree depth is another important parameter in the DT classifier. Thus, an extensive evaluation has been performed with a large range of values of the tree depth. Fig. 8 shows the results of four performance metrics (i.e., Overall accuracy, Precision, Recall, and F1 Score) in the y-axis and the tree depth in the x-axis. The tree obtains the highest F1 score value with the depth of 24. Interestingly, it is consistent between the four metrics.

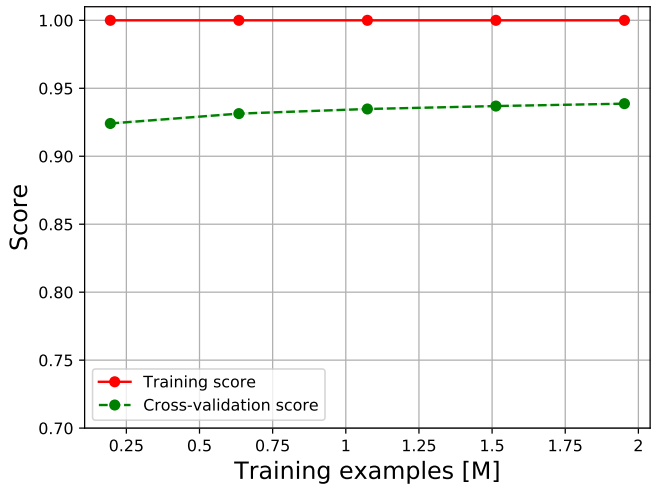


Fig. 6: Learning curve results show the impact of the number of training samples on the overall test accuracy using only 30% of Tranalyzer feature set.

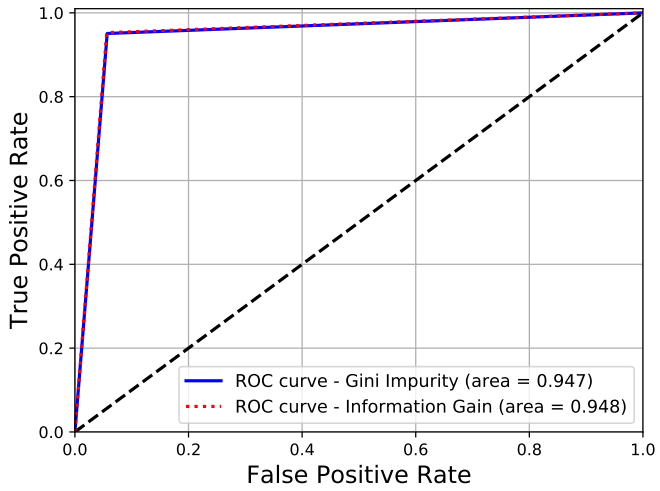


Fig. 7: Gini Impurity versus Information Gain using Tranalyzer feature set.

VI. Conclusion and Future Work

In this work, we proposed an ML-based approach to detect different NAT behaviors using flow level statistics from passive measurements excluding the IP addresses, port numbers, and the application layer information. A large dataset is generated with ground-truth for evaluating the different approaches. We ran extensive evaluations between the four approaches representing the state-of-the-art in the literature and our proposed approach. We evaluated ten different ML classifiers on eight different traffic flow feature sets. Four of these feature sets were extracted from four well-known network traffic analyzers, namely Argus, Netmate, Tstat, and Tranalyzer. The other four feature sets represent the state-of-the-art approaches, A1 to A4, in the literature. These include representations from

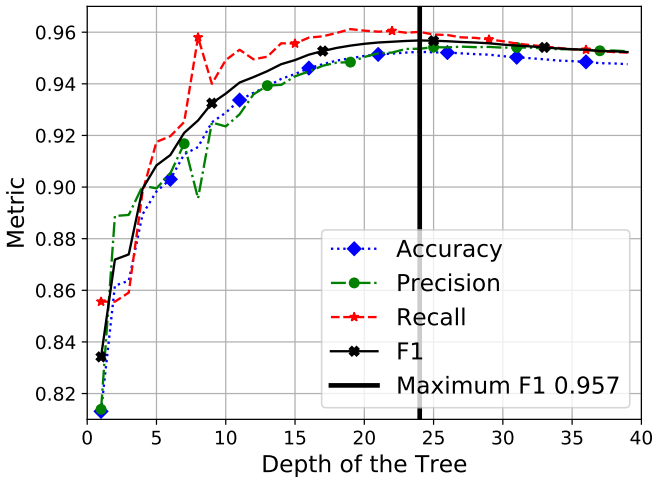


Fig. 8: The impact of the depth of the DT on the accuracy using only 10% of Tranalyzer feature set. The maximum value of F1 Score is obtained at the depth of 24.

other network traffic flow based research [4], [22] to research employing operating system and application layer information [3], [5]. Our results show that the proposed solution using the DT classifier with Tranalyzer feature set without IP addresses, port numbers, operating system and/or application layer information achieves the highest F1 score (overall accuracy) with the minimum computational cost among the approaches evaluated. We further analyzed this solution in terms of parameter sensitivity and feature space. Future work will investigate the application of the proposed solution to other proxy systems for exploring different behaviors.

Acknowledgements

This research is supported by Natural Science and Engineering Research Council of Canada (NSERC) and 2Keys Corp. The research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

References

[1] A. S. Khatouni, M. Mellia, M. A. Marsan, S. Alfredsson, J. Karlsson, A. Brunstrom, O. Alay, A. Lutu, C. Midoglu, and V. Mancuso, "Speedtest-like measurements in 3g/4g networks: The monroe experience," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, pp. 169–177, Sep. 2017.

[2] M. Abu Rajab, F. Monrose, and A. Terzis, "On the impact of dynamic addressing on malware propagation," in *Proceedings of the 4th ACM Workshop on Recurring Malcode*, WORM '06, (New York, NY, USA), pp. 51–56, ACM, Nov. 2006.

[3] G. Maier, F. Schneider, and A. Feldmann, "Nat usage in residential broadband networks," in *Proceedings of the 12th International Conference on Passive and Active Measurement*, PAM'11, (Berlin, Heidelberg), pp. 32–41, Springer-Verlag, Mar 2011.

[4] Y. Gokcen, V. A. Foroushani, and A. N. Z. Heywood, "Can we identify nat behavior by analyzing traffic flows?," in *2014 IEEE Security and Privacy Workshops*, pp. 132–139, May 2014.

[5] T. Komárek, M. Grill, and T. Pevný, "Passive nat detection using http access logs," in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, Dec 2016.

[6] D. J. D. Touch, "Updated Specification of the IPv4 ID Field." RFC 6864, Feb 2013.

[7] A. Lutu, M. Bagnulo, A. Dhamdhere, and K. C. Claffy, "Nat revelio: Detecting nat444 in the isp," in *Passive and Active Measurement* (T. Karagiannis and X. Dimitropoulos, eds.), (Cham), pp. 149–161, Springer International Publishing, Mar. 2016.

[8] A. Müller, F. Wohlfart, and G. Carle, "Analysis and topology-based traversal of cascaded large scale nats," in *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '13, (New York, NY, USA), pp. 43–48, ACM, Dec. 2013.

[9] V. Krmicek, J. Vykopal, and R. Krejci, "Netflow based system for nat detection," in *Proceedings of the 5th International Student Workshop on Emerging Networking Experiments and Technologies*, Co-Next Student Workshop '09, (New York, NY, USA), pp. 23–24, ACM, Dec 2009.

[10] S. Abt, C. Dietz, H. Baier, and S. Petrović, "Passive remote source nat detection using behavior statistics derived from netflow," in *Emerging Management Mechanisms for the Future Internet* (G. Doyen, M. Waldburger, P. Čeleda, A. Sperotto, and B. Stiller, eds.), (Berlin, Heidelberg), pp. 148–159, Springer Berlin Heidelberg, Mar 2013.

[11] N. Paxton and J. Mathews, "Identifying network packets across translational boundaries," in *10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 523–530, Oct 2014.

[12] S. Mongkolluksamee, K. Fukuda, and P. Pongpaibool, "Counting natted hosts by observing tcp/ip field behaviors," in *2012 IEEE International Conference on Communications (ICC)*, pp. 1265–1270, Jun 2012.

[13] E. Bursztein, "Time has something to tell us about network address translation," 2007.

[14] D. A. Borman, R. T. Braden, and V. Jacobson, "TCP Extensions for High Performance." RFC 1323, May 1992.

[15] G. Wicherski, F. Weingarten, and U. Meyer, "Ip agnostic real-time traffic filtering and host identification using tcp timestamps," in *38th Annual IEEE Conference on Local Computer Networks*, pp. 647–654, Oct 2013.

[16] A. Castiglione, A. De Santis, U. Fiore, and F. Palmieri, "Device tracking in private networks via napt log analysis," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 603–608, Jul 2012.

[17] N. V. Verde, G. Ateniese, E. Gabrielli, L. V. Mancini, and A. Spognardi, "No nat'd user left behind: Fingerprinting users behind nat from netflow records alone," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, pp. 218–227, Jun 2014.

[18] R. Weaver, "Visualizing and modeling the scanning behavior of the conficker botnet in the presence of user and network activity," *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 1039–1051, May 2015.

[19] "Argus: the network audit record generation and utilization system." [Online]. Available: https://qosient.com/argus/., Dec 1994.

[20] S. Burschka and B. Dupasquier, "Tranalyzer: Versatile high performance network traffic analyser," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec 2016.

[21] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, P. D. Torino, and D. Rossi, "Experiences of Internet traffic monitoring with tstat," *IEEE Network*, vol. 25, pp. 8–14, May 2011.

[22] A. Safari Khatouni and N. Zincir-Heywood, "Integrating machine learning with off-the-shelf traffic flow features for http/https traffic classification," in *2019 The 24th Symposium on Computers and Communications (ISCC)*, Jun 2019.

[23] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, pp. 77–93, May 2004.