A PRELIMINARY STUDY FOR IDENTIFYING NAT TRAFFIC USING MACHINE LEARNING

by

Yasemin Gokcen

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
April 2014

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

It is shown in the literature that the NAT devices have become a convenient way to hide the identity of malicious behaviors. In this thesis, the aim is to identify the presence of the NAT devices in the network traffic and (if possible) to predict the number of users behind those NAT devices. To this end, I utilize different approaches and evaluate the performance of these approaches under different network environments represented by the availability of different data fields. To achieve this, I propose a machine learning (ML) based approach to detect NAT devices. I evaluate my approach against different passive fingerprinting techniques representing the state-of-the-art in the literature and show that the performance of the proposed ML based approach is very promising even without using any payload (application layer) information.

LIST OF ABBREVIATIONS USED

ACK	Acknowledge
DNS	Domain Name Server
DPI	Deep Packet Inspection
DR	Detection Rate
DSL	Digital Subscriber Line
eAddr	External Address
ePort	External Port
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
iAddr	Internal Address
ID3	Iterative Dichotomiser 3
IP	Internet Protocol
iPort	Internal Port
ISP	Internet Service Provider
LAN	Local Area Network
NAPT	Network Address and Port Translation
NAT	Network Address Translation
OS	Operating System
PC	Personal Computer
QoS	Quality of Service
RFC 1918	A standard; Address Allocation for Private Internets
RFC 2663	A standard; IP Network Address Translator (NAT) Terminology and Considerations

ROC	Receiver Operating Characteristic
SRM	Structural Risk Minimization
SVM	Support Vector Machine
SYN	Synchronize
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TTL	Time to Live
UDP	User Datagram Protocol
WiFi	Wireless Fidelity
ML	Machine Learning

ACKNOWLEDGMENTS

This research is supported by the Canadian Safety and Security Program (CSSP) E-Security grant, and is conducted as part of the Dalhousie NIMS Lab at http://projects.cs.dal.ca/projectx/. The CSSP is led by the Defense Research and Development Canada, Centre for Security Science (CSS) on behalf of the Government of Canada and its partners across all levels of government, response and emergency management organizations, non-governmental agencies, industry and academia.

CHAPTER 1 INTRODUCTION

Usage of Network Address Translation (NAT) devices is very common in any area where interconnection devices such as computers, laptops and mobiles connect to the Internet. While NAT devices are generally used in local area networks (LAN), which include small groups of computers, they can also be used just for one computer. For example, for home networks, most Internet Service Providers (ISP) give WiFi-enabled NAT home gateways to their users. Thus, when users can connect their devices to the Internet, the private IP addresses are hidden on the Internet by encapsulating private IP addresses with a public IP address. NAT gateways modify IP address information in IP packet headers during transition. Basically, NAT allows a single device, such as a router, to act as agent between the Internet and a private network. This means that only a single unique IP address is required to represent an entire group of computers to anything outside their private network.

NATs are used for many reasons such as shortage of IPv4 addresses. Since an address is 4 bytes, the total number of available addresses is 2 to the power of 32, i.e. 4,294,967,296. This represents the number of computers that can be directly connected to the Internet. In practice, the real limit is much smaller for several reasons. Each physical network has to have a unique Network Number comprising some of the bits of the IP address. The rest of the bits are used as a Host Number to uniquely identify each computer on that network. The number of unique Network Numbers that can be assigned on the Internet is therefore much smaller than 4 billion, and it is very unlikely that all of the possible Host Numbers in each Network Number are fully assigned. NAT usage provides one single public IP address for a group of computers and therefore helps to solve some of the addressing related problems.

To be represented with a public IP address on the Internet is more advantageous for users. Since their private IP addresses are not seen on the Internet, it is easier for them to keep their systems anonymous and indirectly secure. For home users, personal information, such as emails, financial details such as credit cards or cheque numbers can be stolen. For business users, it is more dangerous. There is essential company information such as marketing strategies. If these kinds of essential information are stolen or accessed in any way, this may cause major privacy and security problems. For these reasons, companies can use firewall technologies to keep their

systems safe. Firewalls are placed between the user and the Internet and verify all traffic before allowing it to pass through so no unauthorized user would be allowed to access the company's file or email server.

Furthermore, another (alternative) viable solution is NAT. A NAT device is also placed between the user and the Internet and it automatically protects the systems without any special set-up because it only allows connections that are originated on the inside network. For instance, an internal client can connect to an outside FTP server, but an outside client will not be able to connect to an internal FTP server because it would have to originate the connection, and NAT will not allow that. In a NAT network, it is still possible to provide some internal servers available to the outside world by opening inbound ports to specific internal addresses, thus making services such as FTP or Web available in a controlled way.

Moreover, it is easier to manage the large networks for network administrators if there are NATs. That is because NAT divides large networks into smaller ones. There are groups of computers behind NAT devices. All the computers behind a NAT device, are represented with just one IP address to the rest of the internet. Therefore, if any change happens within these computers, such as adding or removing IP addresses, this does not affect the rest.

All these advantages provide NAT usage to be common. However, also because of these reasons NAT technology becomes useful for attackers and users who want to hide their real identities. Hence, NAT usage increases both in legitimate environments and in illegitimate environments. Thus, the number of devices behind a NAT device (gateway) becomes more important to detect to understand the anomalies in a given systems traffic and usage. Furthermore, it is not possible to get information such as private IP addresses, which belong to the devices behind a NAT gateway just by visualizing the traffic. In other words, identifying the NAT gateways and the computers behind such a gateway becomes a very challenging problem.

NAT gateway has at least two interfaces and it has two different IP addresses for each interface; namely internal and external IP addresses. The internal IP address is for communicating with hosts behind the NAT (internal) network, while the external one is for communicating with the

external (outside the NAT network) network. Therefore, if anyone from the external network would analyze the NAT network traffic, the only information he/she would see would be the traffic between the NAT and the external network rather than the hosts behind the NAT on the internal network. On the internal network, there might be many hosts (computers), which are connected to the Internet via the NAT, and these machines might have different services installed. However, these will all be opaque when standard techniques (such as IP address or port number analysis or deep packet inspection) are used to visualize or analyze such traffic (data).Thus, it is necessary to understand whether there is a NAT gateway and the number of hosts behind it for any quality of service or security related analysis of a network/system traffic.

Therefore in this thesis, my objective is to study different approaches for identifying the NAT traffic and evaluate them on different types of data sets to understand their benefits and drawbacks. To this end, one approach I used is based on the analysis of packet level traces together with the HTTP user agent information as studied in [2], passive fingerprinting approach. Indeed, such an approach becomes useful in the presence of the HTTP user agent information. However in the cases where such information is not available or not accessible, I propose to analyze the flow level traffic using machine learning (ML) techniques. In this thesis, predicting NAT behavior by using traffic flows and ML classifiers for forensic analysis (as a passive approach) is my new contribution. In short, I investigate these approaches under both encrypted and non-encrypted traffic conditions.

In the rest of this thesis, Chapter 2 summarizes the existing works in the literature. Chapter 3, discusses the NAT mechanism, the data sets for training and testing, the passive fingerprinting and the machine learning approaches, as well as the features employed in both approaches in detail. Chapter 4, presents the evaluation of the different techniques employed using several performance metrics. Chapter 5 introduces the software tool that is designed and developed for my system which identifies the NAT devices in a given network traffic file and predicts the number of potential users behind such devices. Finally, Chapter 6 draws conclusions and gives directions for the future work.

CHAPTER 2 LITERATURE REVIEW

The identification of NAT devices and the number of end users behind such devices is relatively a new research area. To this end, different algorithms were proposed, but generally, researchers used some form of an active or a passive fingerprinting approach to identify NAT behaviors in network traffic. They analyze certain parameters within the TCP/IP (Transmission Control Protocol/ Internet Protocol) protocol and most of the time evaluate performances on their experimental systems with synthetic NAT data sets.

In the case of passive approaches, Bellovin et al. [8] identified that consecutive packets carry sequential IP Identification (ID) fields, which were included in the IP header and generally were used as counters. Therefore, they concluded that it was possible to count the string values of those IP ID fields to find the number of hosts. However, such an approach might have some complications when faced with packets with zero IP IDs or packets using byte-swapped counters. Moreover, the recent versions of OpenBSD and FreeBSD use pseudo-random number generators for the IP ID fields. So counting such values will not work accurately on such Operating Systems (OSs). Another similar work was proposed by Beverly et al [7], where a classifier was employed to infer the OS passively and to find the number of hosts behind a NAT. Beverly et al used Time-To-Live (TTL), Do-not-fragment (DF), Window-size and SYN-size parameters. Phaal et al [5] took the advantage of IP TTL, while Miller et al [6] analyzed the tcpdump packets. They checked certain fields, namely Type of Service (TOS), total length, IP ID, TTL, source port, window and TCP options in the TCP/IP header to fingerprint different OSs. The idea was that if an IP address had more than one OS associated with it, this could indicate a NAT device with different OSs behind it. However, such an approach will give false alarms if a computer has more than one OS installed on it, even though it is not behind a NAT device.

Moreover, Rui et al. [4] proposed the use of a Support Vector Machine (SVM) based classifier to detect the presence of a NAT in the given traffic traces. They captured traffic on five different hosts that were potential NAT devices. Then they trained a SVM classifier on the captured traffic to be able to detect the NAT device among the five potential hosts. Their traces were limited by eight features; the number of packets sent out, the number of packets received, the number of UDP packets, the number of TCP packets, the number of DNS request packets, the number of

FIN packets, the number of RST packets and the number of SYN packets to represent the character of packet traces. Based on these, they calculated activity values to show activeness of a host. Their hypothesis was that compared to an ordinary host, the hosts behind a NAT device sent more bytes, needed more network connections, visited more web sites, which could mean more DNS requests, and produced more complex traffic traces. Therefore, a host behind a NAT device would have higher activity value than an ordinary one. They labeled their traffic data as ordinary hosts and NAT hosts. Then, they applied binary classification using the SVM. Their experiments showed that their approach was effective on their data sets and the accuracy increased when there were more hosts behind the NAT device. However their approach has some drawbacks. They captured the traffic traces on each host separately to identify if one or more of those hosts are NAT. In practice it may not be possible to collect traffic on the hosts that we are suspicious of being a NAT device, because we may not have control on such devices to collect the data. Moreover, their approach also necessitates access to application level data such as DNS queries, which again may not be possible in practice if the data is encrypted.

Maier et al. [2] focused on detecting DSL lines that use NAT to connect to the Internet. They first aimed to identify whether there were a NAT device, and then to identify the number of users behind that NAT. Their approach is a form of passive fingerprinting and is based on the IP TTL and the HTTP user agent strings. They extracted the OS and the browser family and its versions from the HTTP user agent strings. Indeed, this necessitates deep packet inspection into the payload of a packet. However, they indicated that if they did not have access to the payload information, then they only used the IP TTL information. They analyzed the user agent strings of typical browsers and they ignored the ones, which came from mobile devices and gaming consoles. Their results indicated that they could identify the potential NAT devices in their DSL data sets.

Krmicek et al. [10] proposed to use traffic flow data to analyze the traffic. It should be noted here that network flows are derived from a 5-tuple information consisting of protocol (TCP/UDP), ‘forward’ and ‘backward’ IP addresses and corresponding port numbers, where IP addresses that match within a finite temporal window forms a flow. Given this, they proposed to extend the flow record with three new fields: TTL field, IP ID field and TCP SYN packet size field.

However, they did not report any evaluation results [10]. Their work only includes description of their proposed system. Actually, this description is similar to Maier et al.'s [2] in terms of using the distinct IP addresses and their TTLs. Even though, Krmicek et al.'s proposal seems interesting, the fact that they aim to extend the NetFlow standard with 3 additional fields makes it difficult to be adapted in practice. Probably this is one of the reasons why no implementation or evaluation of it exists in the literature.

In the case of active approaches, Murakami et al. [3] focused on the Medium Access Control (MAC) address of a device and proposed a NAT router, which relayed the MAC address of PCs based on FreeBSD. NAT does not have information about the data link layer because it translates IP addresses in the network layer. So they used two functions; obtaining source MAC address and overwriting an Ethernet header. They added another mechanism by using *pcap* both to obtain MAC addresses and to overwrite *Ethernet* headers. According to their evaluation process, their MAC address relaying NAT router confirmed that a LAN could identify PCs that were behind a NAT. The drawback of this system is that it requires the use of their specific relaying system.

In another active method, Ishikawa et al. [1] proposed to identify the computers behind a NAT gateway with proxy authentication on a proxy server. Their target application was WWW. They used realm field in the authentication header by associating it with the MAC address of a client computer. After the authentication was performed successfully, the realm was shown to the user as a prompt message. Their proposed system requires Java Runtime Environment (JRE) on each client machine. They assumed that a web browser always adds the authentication header to its request message when authentication was completed successfully. However, this may not be the case sometimes. Moreover, this method could only be used for their target application and the proxy conditions require the JRE code the authors developed.

In summary, Maier et al. [2]'s approach, seems to be the best performing approach reported in the literature albeit its requirement for payload information. Thus, in this research, I re-engineered their approach to understand its advantages and disadvantages. In addition, for the cases where payload information is not available or opaque (such as encrypted traffic), I propose

a ML based approach based on the flow based attributes. To this end, my aim is to study whether general enough patterns can be learned automatically to identify the behavior of NAT devices on a network/system that is under analysis. This is in some ways similar to the work in [4]. However, my proposed system does not require traffic traces from potential NAT hosts andI employ flow features rather than packet features without accessing application (payload) level information. To the best of my knowledge, this research is the first one aiming to evaluate these different approaches on different traffic traces from different networks to identify the potential NAT traffic (devices).

CHAPTER 3 METHODOLOGY

In this research, I aimed to evaluate two different approaches on identifying potential NAT devices on given traffic traces. These approaches are: my proposed ML based approach and the passive fingerprinting approach from [2]. For the ML approach, I employ the C4.5 and Naive Bayes learning techniques as my classifiers. The reason I chose these two learning algorithms are two folds. C4.5 learning technique provides the solution it learns from the data in a tree form using if-then-else format. This makes it easy for a human expert to analyze the solution and to understand what the algorithm learned. In other words, the solution is no longer a black box. On the other hand, Naive Bayes is one of the well-known statistical learning algorithms (albeit with an opaque solution) so it naturally represents a standard baseline classifier for this work. As for the passive fingerprinting approach, I re-engineered and employed the algorithm introduced by Maier et al. [2] summarized in section 3.4.

For all approaches, I have used the same data sets namely Nims-NAT and Partner-NAT including both the encrypted and the non-encrypted traffic flows. Furthermore, by using the aforementioned approaches, I aim to analyze the behavior of a NAT device both with and without the payload information to understand how much one gains by having access to the payload, i.e. application level information.

3.1. NAT Overview

As discussed earlier, NAT is the process of modifying IP address information in IP packet headers while in transit across a traffic gateway. It is common to hide an entire IP address space, usually consisting of private IP addresses, behind a single IP address in another address space (usually public). To avoid ambiguity in the handling of returned packets, a one-to-many NAT must alter higher-level information such as TCP/UDP ports in outgoing communications and must maintain a translation table so that returned packets can be correctly translated back for its intended target host. RFC 2663 uses the term NAPT (Network Address and Port Translation) for this type of NAT. Since this is the most common type of NAT, it is often referred to simply as NAT.

The majority of NATs map multiple private hosts to one publicly exposed IP address. In a typical configuration, a local network uses one of the designated "private" IP address subnets (RFC 1918) [24]. A gateway (router) on that network has a private address in that address space. The router is also connected to the Internet with a "public" address assigned by an ISP. As traffic passes from the local (private) network to the Internet, the source address of each packet is translated on the fly from a private address to the public address. The router tracks basic data about each active connection (particularly the destination address and the destination port). When a reply returns to the router, it uses the connection tracking data that it stored during the outbound phase to determine the private address on the internal (private) network to which to forward the packet. Figure 1 shows an example of a general network structure that has a NAT machine.

All Internet packets have a source IP address and a destination IP address. Typically packets passing from the private network to the public network will have their source addresses modified while packets passing from the public network back to the private network will have their destination addresses modified. To avoid ambiguity in how to translate returned packets, further modifications to the packets are required. The vast bulk of Internet traffic is TCP and UDP packets, and for these protocols the port numbers are changed so that the combination of IP and port information on the returned packet can be unambiguously mapped to the corresponding private address and port information. Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort will be sent through eAddr:ePort. Any external host can send packets to iAddr:iPort by sending packets to eAddr:ePort.

For the purpose of this research, first of all, I observed the behavior of the NAT protocol in practice. To this end, I captured packets at both the input and the output sides of an NAT device. I sent and captured packets from a client PC (at a home network) to the web server at our faculty, namely www.dal.ca. Within the home network, the home network router provides a NAT service.

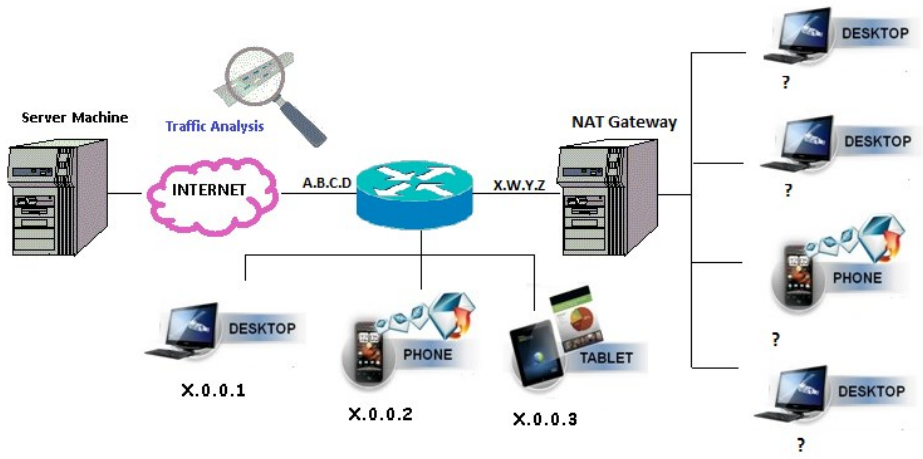


Figure 1 A general NAT network

Figure 2 shows my scenario for capturing traffic to observe the NAT protocol in practice. I have collected a Wireshark trace on the client PC in my home network. It is called the Home_Side trace. I am also interested in the packets being sent by the NAT router to the ISP network, I have collected a second trace file on the ISP network, Figure 2. Client-to-server packets captured by Wireshark [19] at this point would have undergone NAT translation. The Wireshark trace captured on the ISP side of the home router is called the ISP_Side trace.

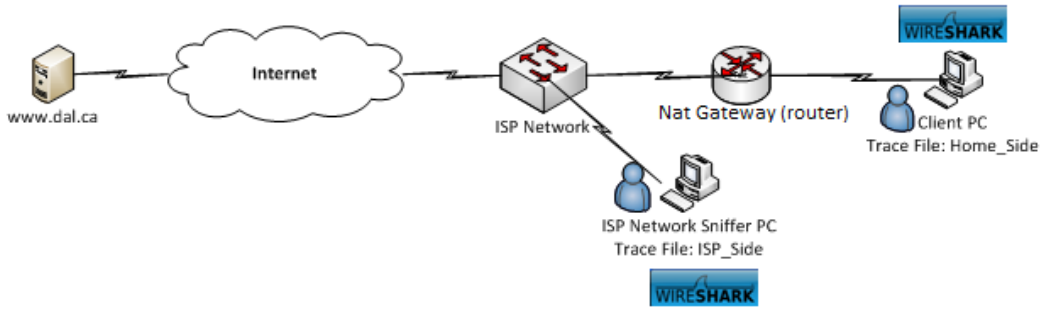


Figure 2 NAT traffic collection scenario

3.1.1. Translation of the End Point

In the following I analyze the Home_Side and ISP_Side traffic traces I captured using Wireshark [19]. Wireshark is an open source protocol analysis tool. NAT usage provides that all

communication that are sent to external hosts actually contain the external IP address and port information of the NAT device instead of the internal host(s) IPs or port numbers.

Figure 3 shows that the HTTP GET sent from the client to the faculty server (whose IP address is 129.173.21.171) at time 0.004374. The following are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET request, highlighted in blue in Figure 3.

Figure 4 shows the packet with *Source IP: 192.168.137.2, Source Port: 1268, Destination IP: 129.173.21.171, Destination Port: 80* at time 0.013089 for the corresponding *200 OK*HTTP message received from the faculty server. The following are the source and destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP *200 OK* message, highlighted in blue in Figure 4. *Source IP: 129.173.21.171, Source Port: 80, Destination IP: 192.168.137.2, Destination Port: 1268.*

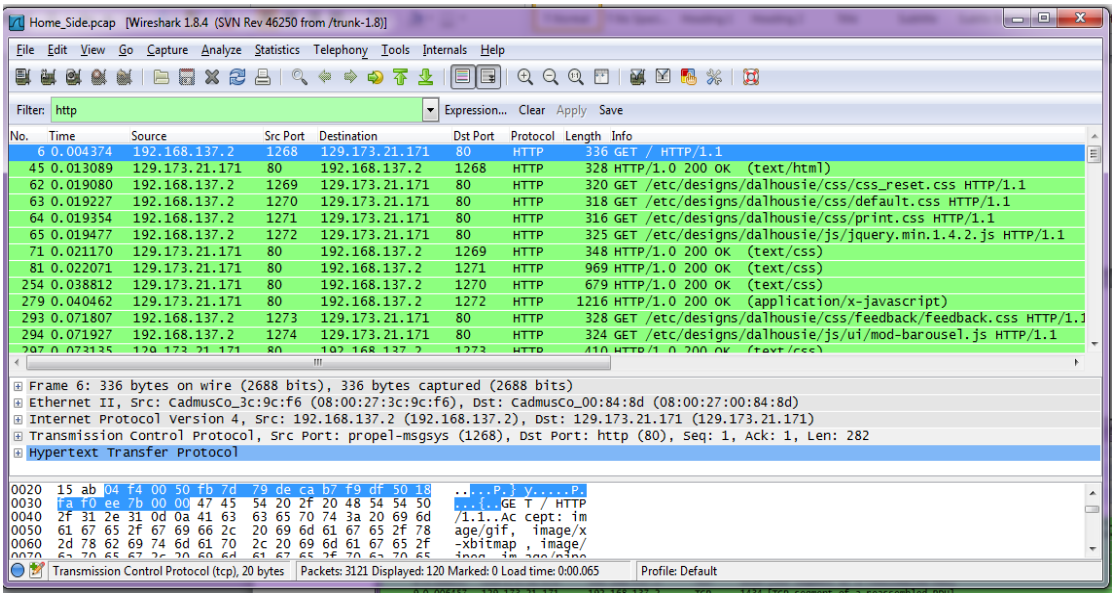


Figure 3 Home-Side trace: Consider the Source IP/Port and the Destination IP/Port for the "HTTP GET" and the "200 OK HTTP" messages

Recall that before a GET command can be sent to an HTTP server, TCP must first set up a connection using the three-way SYN/ACK handshake. Considering Figure 2, you can find the following information for SYN and ACK messages:

SYN Time: 0.002799

SYN Source IP: 192.168.137.2, Source Port: 1268

SYN Destination IP: 129.173.21.171, Destination Port: 80

ACK Time: 0.004032

ACK Source IP: 129.173.21.171, Source Port: 80

ACK Destination IP: 192.168.137.2, Destination Port: 1268

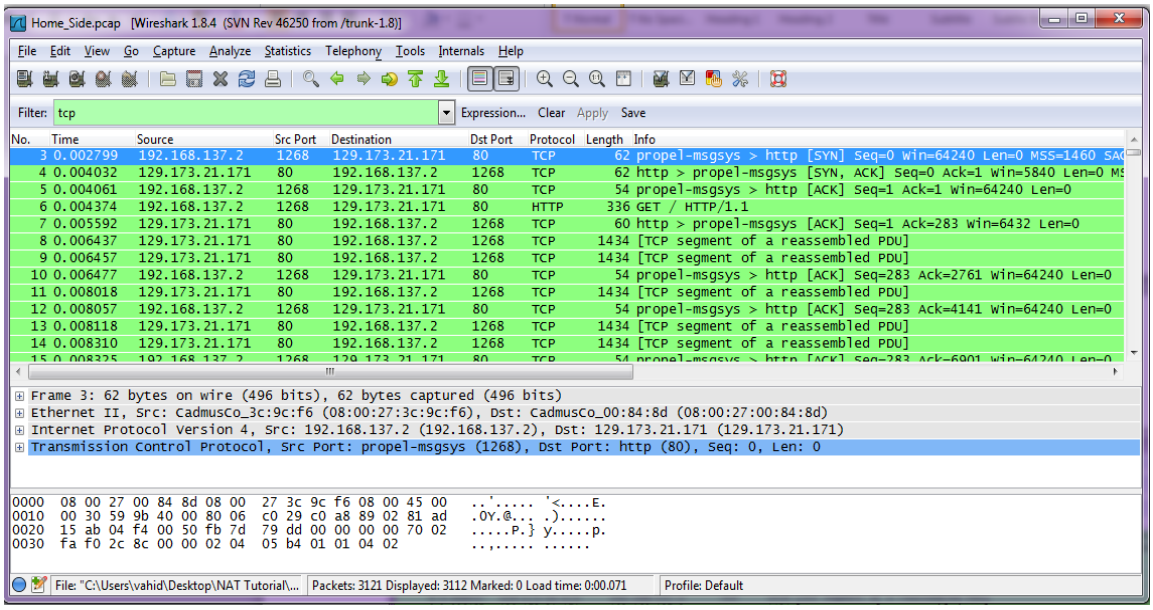


Figure 4 Home-Side trace: Consider the Source IP/Port and the Destination IP/port for the three-way SYN/ACK handshake

In the following, I will focus on the two HTTP messages ("GET" request and "200 OK" reply) and the TCP SYN and ACK segments identified above in the ISP_Side trace captured on the ISP network. Because these captured frames have already been forwarded through the NAT router, I am going to show that some of the IP addresses and port numbers have been changed as a result of the NAT translation.

Note that the time stamps in ISP_Side and Home_Side traces are not synchronized since the packet captures at the two locations shown in Figure 2 were not started simultaneously. Indeed,

you can find that the timestamps of a packet captured at the ISP network is actually different than the timestamp of the packet captured at the client PC.

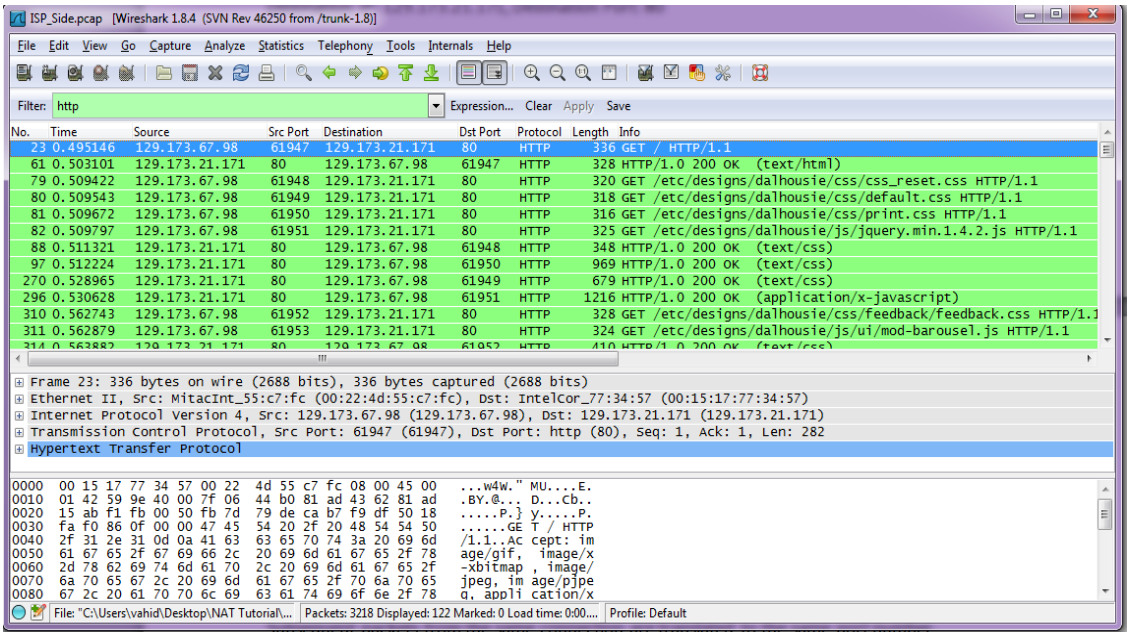


Figure 5 ISP-Side trace: Consider the Source IP/Port and the Destination IP/port for the "HTTP GET" and the "200 OK HTTP" messages

Consider Figure 5, the NAT ISP_side trace, the HTTP GET message has been sent from the client to the faculty server at time 0.495146, highlighted in blue. The source and the destination IP addresses and TCP source and destination ports on the IP datagram carrying this HTTP GET request message are as the following: *Source IP: 129.173.67.98, Source Port: 61947, Destination IP: 129.173.21.171, Destination Port: 80*. As you can see, in comparison to Figure 3, the destination IP and port have not been changed, but the source IP and port have been translated by the NAT router.

My observations show that when a computer on the private (internal) network sends a packet to the external network as expected, the NAT device replaces the internal IP address in the source field of the packet header (sender's address) with the external IP address of the NAT device. NAT may then assign the connection a port number from a pool of available ports, inserting this port number in the source port field (much like the post office box number), and forwards the packet to the external network. The NAT device then makes an entry in a translation table

containing the internal IP address, original source port, and the translated source port. Subsequent packets from the same connection are translated to the same port number. The computer receiving a packet that has undergone the NAT device establishes a connection to the port and IP address specified in the altered packet, oblivious to the fact that the supplied address is being translated (analogous to using a post office box number).

A packet coming from the external network is mapped to a corresponding internal IP address and the port number from the translation table, replacing the external IP address and the port number in the incoming packet header. The packet is then forwarded over the internal network. Otherwise, if the destination port number of the incoming packet is not found in the translation table, the packet is dropped or rejected because the NAT device does not know where to send it.

Most importantly, in addition to the IP address and the Port fields, two more fields in the IP datagram have also been changed, TTL and Checksum. Consider Figure 6 and Figure 7 to compare the TTL and Checksum fields of the HTTP GET message in the Home-Side and the ISP-Side traces.

TTL value can be thought of as an upper bound for the time that an IP datagram can exist on the internet. The TTL field is set by the sender of the datagram, and reduced by every router on the route to its destination. The purpose of the TTL field is to avoid a situation in which an undeliverable datagram keeps circulating on the internet. Under IPv4, every host that passes the datagram must reduce the TTL by one unit. In practice, the TTL field is decreased by one at every hop (router/gateway). However, this might not be the case by every NAT device. In the case of a NAT device, decreasing the TTL by one depends on the policies and the objectives of the organization that sets up the NAT device. Indeed, it is possible to configure a NAT device, as well as any other network router/gateway, to not to decrease the TTL value. In this scenario however, I have observed that the TTL has been decreased by one, Figure 7.

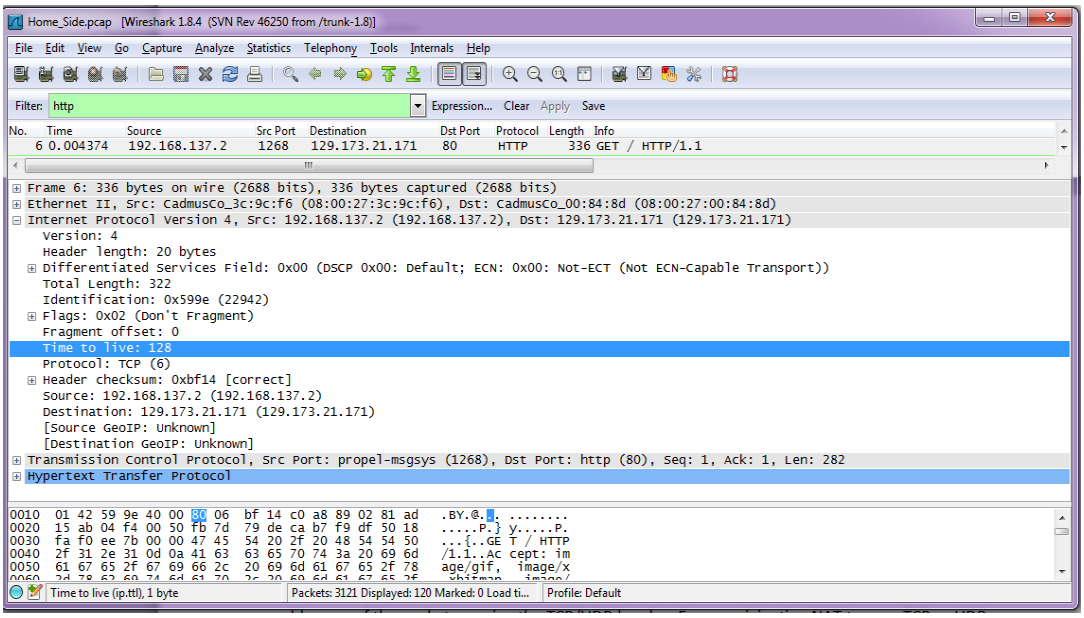


Figure 6 Home-Side trace: Consider the "TTL" and the "Checksum" fields for the "HTTP GET" message

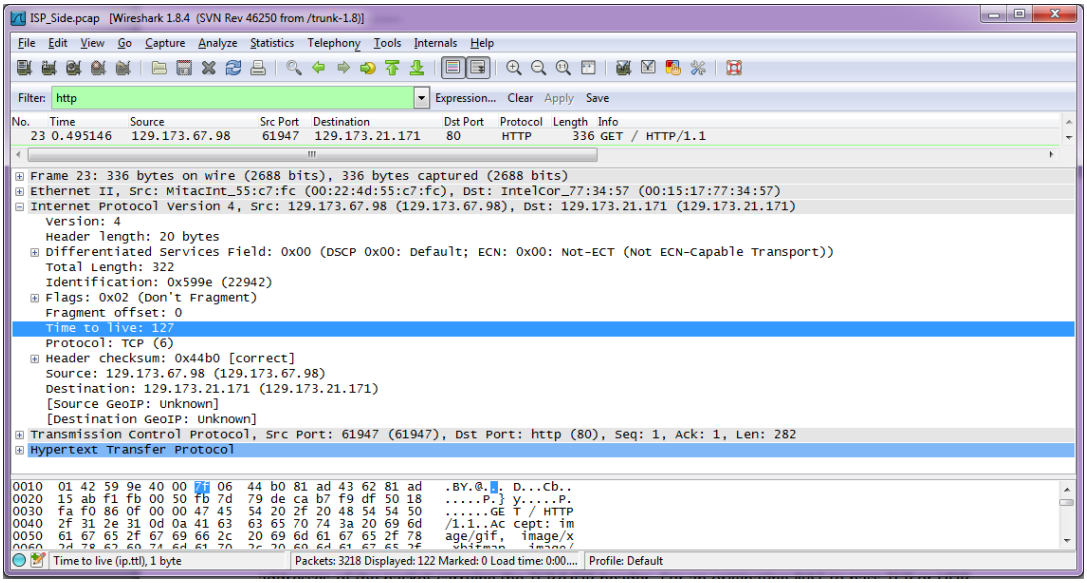


Figure 7 ISP-Side trace: Consider the "TTL" and the "Checksum" fields for the "HTTP GET" message

Regarding the checksum value, the major transport layer protocols, TCP and UDP, have a checksum that covers all the data they carry as well as the TCP/UDP header plus a "pseudo-header" that contains the source and destination IP addresses of the packet carrying the TCP/UDP header. For a sending NAT device to forward the TCP or the UDP packets successfully, it must re-compute the TCP/UDP header checksum based on the translated IP

addresses, not the original ones, and put that checksum into the TCP/UDP header of the first packet of the fragmented set of packets. The receiving NAT must re-compute the IP checksum on every packet it forwards to the destination host, and also recognize and re-compute the TCP/UDP header using the retranslated addresses. In the ISP_Side trace file, Figure 8, you can find the following information for the three-way SYN/ACK handshake:

SYN Time: 0.493603

SYN Source IP: 129.173.67.98, Source Port: 61947

SYN Destination IP: 129.173.21.171, Destination Port: 80

ACK Time: 0.494453

ACK Source IP: 129.173.21.171, Source Port: 80

ACK Destination IP: 129.173.67.98, Destination Port: 61947

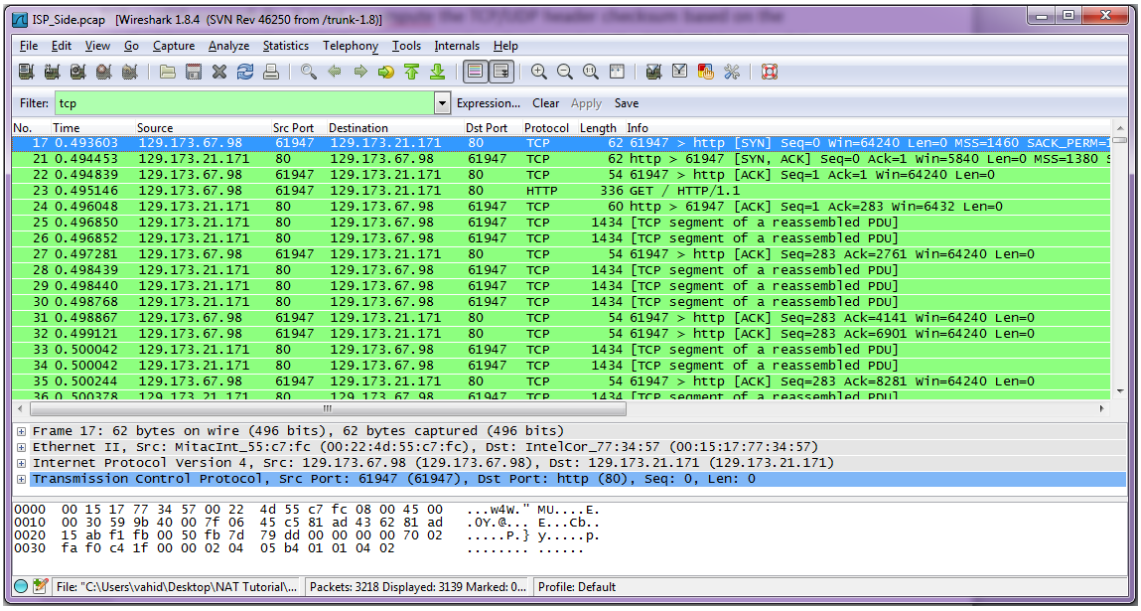


Figure 8 ISP-Side trace: Consider the Source IP/Port and the Destination IP/port for the three-way SYN/ACK handshake

Comparing Figure 8 and Figure 4, you can see that the source IP and the port in the ACK message (direction from the home side to the ISP side) and the destination IP and the port in the SYN message (direction from the ISP side to the home side) have been translated by the NAT router.

3.1.2. Visibility of NAT Operations

Typically the internal host (the host in the private network) is aware of the true IP address and the TCP/UDP port of the external host (the host in the public network), unless the external host is also behind a NAT device. The NAT device may function as the default gateway for the internal host. However, the external host is only aware of the public IP address for the NAT device and the particular port being used to communicate on behalf of a specific internal host.

As discussed above, NAT only translates the IP addresses and the ports of its internal hosts, possibly decreasing the TTL value by one, and re-computing the checksum. This results in translating the IP addresses and the ports of its internal hosts to hide the real address of an internal host on a private network. Because the internal addresses are all disguised behind one publicly accessible address, it is impossible for the external hosts to differentiate between the traffic originated from a host behind a NAT from a host that is not behind a NAT. As a result, these networks (hosts behind a NAT device) are ideal for attackers to hide their identities. If an attacker hides his/her identity behind a NAT device, it is very difficult to find the exact attacker node.
Thus, a mechanism of identifying the NAT traffic is anticipated to be very useful, as the attackers behind the NAT devices can easily violate the network security. This is my main motivation in this research. However achieving this is not possible by using the typical network traffic analysis techniques such as Wireshark type tools. In short, I use a combination of techniques including ML algorithms to achieve this goal.

3.2. Data Sets Employed

In this research, two different data sets from two different organizations (networks), namely Nims-NAT and Partner-NAT, are employed to evaluate the aforementioned approaches. These are traffic data sets in the form of *tcpdump* log files without any payload information. Moreover, Nims-NAT also includes the web access log files, which belong to the same time period. Nims-NAT data is collected over a week in November 2012 including both encrypted and non-encrypted traffic.

In total, there are 177,493flows in Nims-NAT dataset. All these flows were matched with the web access log data files which include both encrypted (HTTPS) and non-encrypted (HTTP) data. The flows in these data sets are labeled into two categories: (i) NAT flows; and (ii) OTHER flows. It should be noted here that a partial ground truth, in terms of NAT devices, is known for these data sets. This means the minimum number of NAT devices is known but not the maximum. In all of the Nims-NAT data sets, there are 95 different OSs and 105 different browser families. The detailed list of OSs and browsers observed in these data sets are presented in section 4.1.

As for detailing the "partial" ground truth regarding the NAT devices in this data set, I do not know every NAT device that accesses the web server where I captured the traffic. I only know the NAT devices of our labs. Indeed, computers from these labs (behind the NAT device) access the web server (where the data sets are collected). Thus, the choice of these data sets enables me to have partial ground truth information about the presence of NAT devices in the Nims-NAT data set. Indeed, there could be more NAT device traffic in the data sets given that other hosts outside of the labs access the web server too. However, this cannot be known for sure, hence the number of hosts known is the minimum but not the maximum.

Second data set is provided to me with the ground truth as well (in terms of NAT flows vs OTHER flows) by a medium sized private company. I will refer to this data set as Partner-NAT hereafter. Given the privacy issues related to this data set, I will not be able to provide any further details about the Partner-NAT.

Each of these data sets is labeled in two ways depending on whether the data is encrypted or not. They are labeled as (i) NAT flows and (ii) OTHER flows when there is only unencrypted traffic and they are labeled as (i) Unencrypted-NAT flows, (ii) Encrypted-NAT flows and (iii) OTHER flows when there is both encrypted and unencrypted traffic, where the ground truth about the NAT devices is known. Therefore I have two different classifications; binary classification and three-way classification for both data sets.

For the Nims-NAT data set, there is one NAT device in our labs. Therefore, the flows and packets with this IP address were all labeled as NAT. All the rest was labeled as OTHER (meaning non-NAT). Hereafter, I will refer to those flows as NAT flows and OTHER flows. As a result of this, there are 12,168 NAT flows among all the flows (in total 177,493 flows) in the data set. It means that about 7% of this data set is NAT traffic. Please note that for my proposed approach where I evaluated different learning techniques (C4.5 and Naïve Bayes) to automatically identify NAT traffic, I removed the IP addresses and the port numbers (flows do not have payloads) from the flows.

For the Partner-NAT data set, which was given to me with labels, my analysis of the IP addresses in that data set show that there are 3 IP addresses that belong to NAT devices. However, because of the privacy reasons, I am not allowed to declare those in this thesis.

Furthermore, for the Nims-NAT data set, I have also employed the HTTP and HTTPS web access log files for my research. To this end, I match them with the captured traffic files, which are in *tcpdump* format, based on time stamps. HTTP web access log files include unencrypted data while HTTPS include encrypted data. On the other hand, all the NAT traffic in the Partner-NAT data set consist of encrypted data. It should be noted here that there are no corresponding HTTP and HTTPS data sets in the case of the Partner-NAT data set. Finally, my analysis of the *tcpdump* data sets from both networks show that whenever the label is encrypted, that traffic is either on port 443 or port 22. These are the standard Secure Socket Layer (SSL) and Secure Shell (SSH) protocol ports, respectively.

There are only 212 flows which are labeled as Encrypted-NAT in the Nims-NAT data set among 12,168 NAT flows, while all the NAT flows are unencrypted (labeled as Unencrypted-NAT) in the Partner-NAT data set. Table 1 shows the statistics for both data sets. Tables 2 and Table 3 show the number of flows in training and testing data sets for both Nims-NAT and Partner-NAT data sets.

Table 1 The Number of Flows in the Traffic Data Employed in This Research

		Number of Flows			
		Encrypted-NAT	**Unencrypted-NAT**	**OTHER**	**TOTAL**
Data Sets	**Nims-NAT**	212	11956	165325	177493
	Partner-NAT	0	99242	44474	143716

Table 2 The Number of Flows in the Training and the Testing Data Sets in the Unencrypted Traffic

			Number of Flows		
			NAT	**OTHER**	**TOTAL**
Data Sets	**Nims-NAT**	**Training**	9126	9126	18252
		Testing	3042	156199	159241
		Total	12168	165325	177493
	Partner-NAT	**Training**	9126	9126	18252
		Testing	90116	35348	125464
		Total	99242	44474	143716

Table 3 The Number of Flows in the Training and the Testing Data Sets in the Encrypted Traffic

			Number of Flows			
			Unencrypted-NAT	**Encrypted-NAT**	**OTHER**	**TOTAL**
Data Sets	**Nims-NAT**	**Training**	159	8967	123993	133119
		Testing	53	2989	41332	44374
		Total	212	11956	165325	177493

3.3. Features Employed

In this thesis, Passive fingerprinting approach employs packet header and payload based features. However, the proposed ML based approach employs traffic flow based features. For the proposed ML based approach, I converted packet based traffic traces (tcpdump files) to flow based data sets. To this end, NetMate [15] and Tranalyzer [20] open source tools are employed to generate the flows and compute the features. The reason I used two different flow generation tools are to check whether the performance of the ML techniques would change from one tool to the other. Given that each of these tools extract different features from a flow, my hypothesis is that this might have an effect on the performance of my proposed ML based approach. It should be noted here that, I do not use the source and destination IP addresses as well as the source and destination port numbers in my feature set to represent the flow traffic to the ML techniques. I think that such information can bias the results. It is well known that port numbers can be assigned dynamically and IP addresses can be spoofed very easily. One can say that in some ways, NATs and proxies are already doing this for free. Thus, my aim here is to find patterns (in other words signatures) to identify NAT traffic automatically without using any biased features. Indeed, to be able to apply my approach both to the encrypted and to the non- encrypted traffic, I do not employ any payload (application layer) information to classifiers in my proposed ML based approach. However, I do employ payload (application layer) information for the passive OS fingerprinting approach that I developed based on Mailer et al.'s approach as they described in [2].This will enable me to understand how much performance gain (if any) could be achieved by employing such information. In the following, I discuss the features employed in this research in more detail.

3.3.1. Features for the ML Based Approach - Netmate Flow Features

NetMate [15] is an open source flow generator. In this case, flows are bidirectional and the first packet of the flow identified by Netmate determines the forward direction. A flow can be uniquely identified by five parameters within a certain time period. These parameters are the source and the destination IP addresses, the source and the destination port numbers and the layer 4 protocol used (TCP/UDP/ICMP). Netmate considers only the UDP and the TCP flows. Moreover, the UDP flows are terminated by a flow timeout, whereas the TCP flows are

terminated upon proper connection teardown or by a flow timeout, whichever occurs first. The flow timeout value employed in this work is 600 seconds as recommended by the IETF [16]. The Netmate features that I used in my experiments are shown in Table 2.

3.3.2. Features for the ML Based Approach - Tranalyzer Flow Features

Tranalyzer [20] is another open source flow generator tool. Similar to Netmate, Tranalyzer generates key parameters and statistics from IP traces that are either live-captured from Ethernet interfaces or achieved as pcap files. In this research, I employed Tranalyzer as a different flow generator because it provides a different feature set than Netmate to represent the flows. While Netmate generates 44 features, Tranalyzer generates 93features. However some of the Tranalyzer features are symbols such as "-, *, /" that are not suitable for the ML based approach. The reason behind this is that some of the ML techniques employ only numeric features. Therefore I can use 76 flow features out of the 93 Tranalyzer provides. Table 3 shows the Tranalyzer features which are employed in this work.

3.3.3. Features for the Passive Fingerprinting Approach

As discussed earlier, I re-engineered the passive fingerprinting approach of Maier et al. [2] to use in this work as a representative of the state-of-the-art systems. In this approach, certain features are used to identify a NAT device. Some of these features require access to the payload. Other features do not have that requirement. I detail these features below.

3.3.3.1. Packet Header Based Features - Time to Live (TTL)

It is known that networking stacks of OSs use well-defined initial IP TTL values (ttl_{init}) in outgoing packets. For instance, Windows uses 128, MacOS uses 64 and Debian based systems use 64 as their TTLs. The TTL field of the IP header is defined to be a timer limiting the lifetime of a datagram. It is an 8-bit field and the units are in seconds. Each router (or other devices) that performs some form of packet forwarding usually decreases the TTL by one (at least), even if the elapsed time was much less than a second. Since this is very often the case, the TTL is effectively a hop count limit on how far a datagram can propagate through the Internet as it is shown in Figure 9 [18].When a router forwards a packet, it is recommended to reduce the TTL by one. If it holds a packet for more than one second, it may decrement the TTL by one for each

second. Therefore, it is anticipated that if there is a NAT box routing in the network, it will decrement the TTL values for each packet that it forwards. However, it is possible for a NAT box not to decrease TTL values. Moreover, users could reconfigure their systems to use a different TTL value.

However, assuming that TTL values are not modified or hidden, these TTL values in the packets could provide some information to infer the presence of a NAT device. If the TTL is ttl_{init}-1, this means that the sending host is directly connected to the Internet, so the monitoring point is one hop away from the host. If the TTL is ttl_{init}-2 then there is a routing device such as a NAT gateway.

A NAT gateway can be a dedicated gateway such as a home router or it can be a regular desktop host. A dedicated NAT gateway will often directly interact with the Internet services, e.g., by serving as a DNS resolver for the local (private) network or for synchronizing its time with NTP servers. It should be noted here that in our datasets, I cannot see any DNS records originated by the known NAT devices.

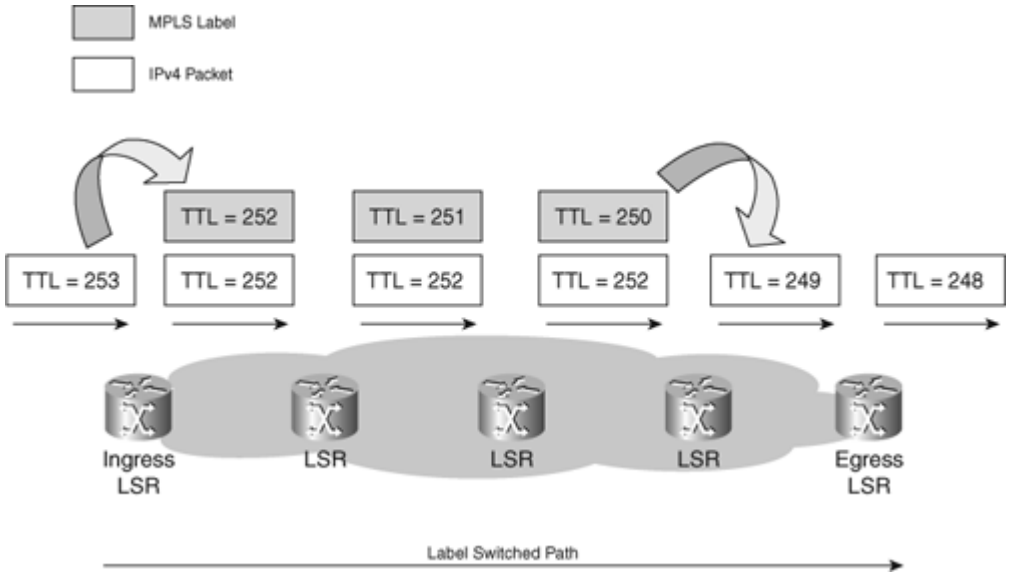


Figure 9 An Example of a Propagation Behavior of TTL [18]

Table 4 Netmate Flow Based Features Employed

No	Feature Name	Abbreviation
1	The protocol (i.e. TCP=6, UDP=17)	proto
2	Total packets in the forward direction	total_fpackets
3	Total bytes in the forward direction	total_fvolume
4	Total packets in the backward direction	total_bpackets
5	Total bytes in the backward direction	total_bvolume
6	The size of the smallest packets sent in the forward direction	min_fpktl
7	The mean size of packets sent in the forward direction	mean_fpktl
8	The size of the largest packet sent in the forward direction	max_fpktl
9	The standard deviation from the mean of the packets sent in the forward direction	std_fpktl
10	The size of the smallest packet sent in the backward direction	min_bpktl
11	The mean size of the packet sent in the backward direction	mean_bpktl
12	The size of the largest packet sent in the backward direction	max_bpktl
13	The standard deviation from the mean of the packets sent in the backward dire	std_bpktl
14	The minimum amount of time between two packets sent in the forward direction	min_fiat
15	The mean amount of time between two packets sent in the forward direction	mean_fiat
16	The maximum amount of time between two packets sent in the forward direction	max_fiat
17	The standard deviation from the mean amount of time between two packets sent in the	std_fiat
18	The minimum amount of time between two packets sent in the backward direction	min_biat
19	The mean amount of time between two packets sent in the backward direction	mean_biat
20	The maximum amount of time between two packets sent in the backward direction	max_biat
21	The standard deviation from the mean amount of time between two packets sent in the	std_biat
22	The duration of the flow	duration
23	The minimum amount of time that the flow was active before going idle	min_active
24	The mean amount of time that the flow was active before going idle	mean_active
25	The max amount of time that the flow was active before going idle	max_active
26	The standard deviation from the mean amount of time that the flow was active before going	std_active
27	The minimum time a flow was idle before becoming idle	min_idle
28	The mean time a flow was idle before becoming idle	mean_idle
29	The maximum time a flow was idle before becoming idle	max_idle
30	The standard deviation from the mean amount of time a flow was idle before becoming idle	std_idle
31	The average number of packets in a sub flow in the forward direction	sflow_fpackets
32	The average number of bytes in a sub flow in the forward direction	sflow_fbytes
33	The average number of packets in a sub flow in the backward direction	sflow_bpackets
34	The average number of bytes in a sub flow in the backward direction	sflow_bbytes
35	The number of times the PSH flag was set in packets travelling in the forward direction (0	fpsh_cnt
36	The number of times the PSH flag was set in packets travelling in the backward direction (0	bpsh_cnt
37	The number of times the URG flag was set in packets travelling in the forward direction (0	furg_cnt
38	The number of times the URG flag was set in packets travelling in the backward direction (0	burg_cnt
39	The total bytes used for headers in the forward direction	total_fhlen
40	The total bytes used for headers in the backward direction	total_bhlen

Table 5 Tranalyzer Flow Based Features Employed

No	Feature Name	Abbreviation
1	Flow Direction	%dir
2	Flow index	flowind
3	System time of first packet	UnixTimeFirst
4	System time of last packet	UnixTimeLast
5	Flow Duration	Duration
6	Ether VlanID	ETHVlanID
7	Layer 4 protocol	Layer4Proto
8	Number of transmitted packets	numPktsSnt
9	Number of Received Packets	numPktsRcvd
10	Number of Transmitted Bytes	numBytesSnt
11	Number of Received Bytes	numBytesRcvd
12	Minimum layer 3 packet size	minPktSz
13	Maximum layer 3 packet size	maxPktSz
14	Average packet load ratio	avePktSize
15	Send packets per second	pktps
16	Send bytes per second	bytps
17	Packet stream asymmetry	pktAsm
18	Byte stream asymmetry	bytAsm
19	IP Minimum delta IP ID	ipMindIPID
20	IP Maximum delta IP ID	ipMaxdIPID
21	IP Minimum TTL	ipMinTTL
22	IP Maximum TTL	ipMaxTTL
23	IP TTL change count	ipTTLChg
24	TCP packet seq count	tcpPSeqCnt
25	TCP sent seq diff bytes	tcpSeqSntBytes
26	TCP sequence number fault count	tcpSeqFaultCnt
27	TCP packet ack count	tcpPAckCnt
28	TCP flawless ack received bytes	tcpFlwLssAckRcvdBytes
29	TCP ack number fault count	tcpAckFaultCnt
30	TCP initial window size	tcpInitWinSz
31	TCP average window size	tcpAveWinSz
32	TCP minimum window size	tcpMinWinSz
33	TCP maximum window size	tcpMaxWinSz
34	TCP window size change down count	tcpWinSzDwnCnt
35	TCP window size change up count	tcpWinSzUpCnt
36	TCP window size direction change count	tcpWinSzChgDirCnt
37	TCP options packet count	tcpOptPktCnt
38	TCP options count	tcpOptCnt
39	TCP aggregated options	tcpAggrOptions
40	TCP maximum segment length	tcpMSS
41	TCP window size	tcpWS

42	TCP Trip timrSyn,Syn-Ack| Syn-Ack, Ack	tcpS-SA/SA-ATrip
43	TCP Round Trip Time Syn,Syn-Ack, Ack | TCP Ack-Ack RTT	tcpRTTSseqAA
44	TCPAck Trip Min	tcpRTTAckTripMin
45	TCP Ack Trip Max	tcpRTTAckTripMax
46	TCP Ack Trip Average	tcpRTTAckTripAve
47	ICMP Echo reply/request success ratio	icmpEchoSuccRatio
48	Number of connections from source IP to different host	connSrc
49	Number of connections from destination IP to different host	connDst
50	Number of connections between source IP and Destination IP	connSrcDst
51	Minimum packet length	MinPl
52	Maximum packet length	MaxPl
53	Mean packet length	MeanPl
54	Lower quartile of packet lengths	LowQuartilePl
55	Median of packet lengths	MedianPl
56	Upper quartile of packet lengths	UppQuartilePl
57	Inter quartile distance of packet lengths	IqdPl
58	Mode of packet lengths	ModePl
59	Range of packet lengths	RangePl
60	Standard deviation of packet lenghts	StdPl
61	Robust stndard deviation of packet lenghts	StdrobPl
62	Skewness of packet lengths	SkewPl
63	Excess of packet lengths	ExcPl
64	Minimum inter arrival time	MinIat
65	Maximum inter arrival time	MaxIat
66	Mean inter arrival time	MeanIat
67	Lower quartile of inter arrival times	LowQuartileIat
68	Median inter arrival times	MedianIat
69	Upper quartile of inter arrival times	UppQuartileIat
70	Inter quartile distance of inter arrival times	IqdIat
71	Mode of inter arrival times	ModeIat
72	Range of inter arrival times	RangeIat
73	Standard deviation of inter arrival times	StdIat
74	Robust standard deviation of inter arrival times	RobStdIat
75	Skewness of inter arrival times	SkewIat
76	Excess of inter arrival times	ExcIat

3.3.3.2. Payload Based Features - HTTP User Agent String

The user agent string identifies the browser that is used to access the web. When a user visits a webpage, his/her browser sends the user-agent string to the server hosting the site that is visited. This string indicates, which browser is used, its version number, and details about the host sending the request, such as the OS and its version.

As in Maier et al.'s work, I parsed the HTTP log files and analyzed the user agent strings as I showed in Table 6 below. I extracted the OS and the browser information from these strings to estimate a lower bound for the number of hosts behind a NAT gateway. Maier et al. [2] limited their analysis to user agent strings from typical browsers such as Firefox, Internet Explorer, Safari and Opera.

However, I did not limit myself with the typical browsers, because in my data sets, I also observed many user agent strings from Android based devices, iPhones and iPads. Therefore, I took them into consideration in my research. I will explain this in more detail in the following sub-sections.

Table 6 An Example for parsing one record (row) of a web access log file

IP Address	1.1.1.1 (Anonymized)
Logname	-
User	-
Date	[06/Nov/2012:16:19:35 --0400]
Request	GET /somedirectory/somefile(anonymized)?time=1352233163159 HTTP/1.1
Code	200
Size	56063
Country	Canada
Referer	http://someserver/somedirectory/somefile(anonymized)
User Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/534.53.11 (KHTML like Gecko) WebClip/7534.55.2 Safari/7534.55.3

3.4. State-of-the-art Representative --Passive Fingerprinting Approach

I evaluated Maier et al.'s approach in four different steps based on the utilization of the features to better understand their effect on the results.

3.4.1. TTL Range

In the simplest form, Maier et al. infer the presence of a NAT device based on the TTL values of packets sent by users. As discussed earlier, they assume that if the TTL is $ttl_{init} - 1$ the sending host is directly connected to the Internet (as the monitoring point is one hop away from the device on which the traffic is monitored (analyzed)). If the TTL is $ttl_{init} - 2$ then there is a routing device (i.e., a NAT device) in the users' premises. In the following, I call the detection of a NAT device based on just the TTL value range as "L1".

Although L1 technique can be used to detect the presence of a NAT device for some networks (aforementioned conditions), it also has some limitations. These may prevent the detection of a NAT device under the following conditions:

- L1 assumes that the number of hops between the machine that the traffic is captured and the machine where the analysis is made, is known. Only then the TTL values used can be interpreted accurately to detect a NAT device. Otherwise, it becomes a problem if I want to analyze any captured traffic from a different network using this where I do not know where/how the traffic is captured.

- L1 assumes that the NAT devices decrease the TTLs for each packet that is forwarded. However some NAT implementations might not decrease the TTL values for some reason or another such as hiding the network topology.

- In Maier et al.'s work, the data set employed was from an ISP so the analysis (monitoring) is performed on the residential users' traffic of the ISP network. In that case, the traffic coming from the residential ISP users naturally goes through a device, which performs the NAT as well as the DNS services. They classify the traffic as coming from a NAT device, if an IP address with a special TTL value specific to DNS packets), later on also has another TTL value in the traffic. Indeed, such a classification is valid only if the DNS and NAT servers are on the same host. However, in some organizations, the DNS and the NAT services might not be on the same host. Therefore, in such cases, the NAT classification will not be correct as the ones seen in Maier et al.'s data sets.

3.4.2. TTL Range and the Distinct TTL Value Per IP Address

In this case, not only the distinct TTL values are observed to detect a NAT device but also the number of distinct TTL values per a distinct IP address are observed to detect NAT traffic. Given that Windows uses a ttl_{init} of 128, MacOS X and Linux use 64, and these ranges are far enough apart to distinguish between them. Thus, observed TTL values can be used to distinguish between Windows and non-Windows OSs. To this end, Maier et al. assume that if more than one TTL value range is observed for one IP address, then that IP address may belong to a NAT device. In the following, I call the detection of a NAT device based on the number of the distinct TTL value ranges per IP address as "L2".

Although utilizing this technique may give a better performance than L1 for detecting a NAT device, it still has some limitations as the following:

- Similar to L1, if the users reconfigure their systems to use a different TTL policy, then this system cannot infer the presence of a NAT device based on the different TTL values.

Table 7 Features Employed for the Passive Fingerprinting Approaches

	From Packet Header		From HTTP User Agent		
Approach	TTL	IP	OS	Browser Family	Browser Version
L1	61	-	-	-	-
L2	61	129.173.13.94	-	-	-
L3	125	129.173.13.94	Windows NT 5.1	-	-
L4	125	129.173.13.94	Windows NT 5.1	Firefox	3.0.3

- When there are two completely different OSs (ex. Linux and Windows) on the same host, this approach would detect two different TTL values (e.g. 64 and 128). So it would classify them as two separate hosts and would infer that there is a NAT device in this traffic, even though there is not.

3.4.3. TTL Range, the Distinct TTL Values Per IP Address, and the Different OS Information in the HTTP User Agent Strings

Given the above limitations and therefore, the false alarms they may cause, Maier et al. extended their technique into the HTTP user agent strings (when the information is available) to observe the OS types and their versions. Then, they anticipate that it is possible to detect a NAT device more accurately based on the OS fingerprint. In the following, I call the detection of a NAT device based on the TTL value range, the number of distinct TTL value ranges per IP address, and the different types of OS information in the HTTP user agent strings as "L3".

However, this technique has the following limitations:

- If all the hosts in a NAT network use the same type of OS, this technique cannot detect the NAT device.

- When there are two versions of an OS on the same host (e.g. Windows XP and Windows 7), this technique would detect one TTL value but two different OS versions. So it would classify them as two separate hosts and would infer that there is a NAT device, even though there is not.

3.4.4. TTL Range, the Distinct TTL Values Per IP Address, the Different OS and the Browser Information in the HTTP User Agent Strings

In this case, in order to have more accurate results, the browser type and version are also extracted from the HTTP user agent string to detect a NAT device. Using this technique, Maier et al. aim to minimize the false alarms that may arise from one host having two different versions of the same OS. The idea behind this technique is that one host might have two different web browsers, but it cannot have two different versions of the same web browser working simultaneously. In the following, I call the detection of a NAT device based on the TTL value ranges per IP address, the number of distinct TTL value ranges, the different type of OSs in the HTTP user agent string per IP address, and the Internet browser information in the HTTP user agent strings per IP address as "L4".

Table 8 Features Employed for the Passive Fingerprinting Approach

	From Packet Header		From HTTP User Agent		
	TTL	Protocol	OS	Browser	Version
Sample Values	54	80/HTTP	Intel Mac OS X 10.5	Firefox	3.05
	54	80/HTTP	Windows NT 5.1	Internet Explorer	6.0
	54	80/HTTP	Windows NT 5.1	Firefox	2.0
	54	80/HTTP	Windows NT 5.1	Firefox	3.0.3
	54	80/HTTP	Windows NT 6.0	Internet Explorer	7.0
	54	80/HTTP	Windows NT 6.0	Firefox	3.0.5

For example, in Table 9, TTL values are the same for each packet. However, there are different OSs such as Windows NT 5.1, Windows NT 6.0 and Intel Mac OS X 10.5. The packets with Windows NT 5.1 OSs have their browsers from the same family, Firefox. Their TTLs, OSs and browser families are the same. According to the previous approach, I could classify these packets as belonging to the same host. However, if I take into consideration the version of the browsers, then the classification could be different. In this example, they use different Firefox versions, so that means using Maier et al.'s assumption above, I classify these packets as belonging to two distinct hosts and therefore, detect that there is a NAT device.

Table 9 Packet Header based features employed, * Normalized by log

	From Packet Header		From HTTP User Agent		
	TTL	Protocol	OS	Browser	Version
Sample Values	56	80/HTTP	Intel Mac OS X 10_8_2	Safari	6.0.2
	119	80/HTTP	Windows NT 6.2	Google Chrome	22.0
	120	80/HTTP	Windows NT 6.2	Google Chrome	22.0

Table 10 Packet Header based features employed, * Normalized by log

	From Packet Header		From HTTP User Agent		
	TTL	Protocol	OS	Browser	Version
Sample Values	56	80/HTTP	Intel Mac OS X 10_7_5	Safari	6.0.1
	119	80/HTTP	Windows NT 6.1	Firefox	11.0
	119	80/HTTP	Windows NT 6.1	Internet Explorer	9.0

Table 11 Packet Header based features employed, * Normalized by log

	From Packet Header		From HTTP User Agent		
	TTL	Protocol	OS	Browser	Version
Sample Values	48	80/HTTP	Android 2.2.x Froyo	Safari	6.0.1
	48	80/HTTP	Android 2.0/1 Eclair	Firefox	11.0
	48	80/HTTP	Windows NT 6.1	Internet Explorer	7.0

In this work, I also use the packets that belong to the Android and other mobile devices, while such devices were not taken into consideration by Maier et al. [2].In my data sets, I observed many records, which belonged to the mobile devices. There are many types of Android devices, iPhone devices and iPad devices. When I analyze their user agent strings, I can also see the device models; such as Nessus One or Samsung SGH-1896.As presented in Table 12, in this case, using the aforementioned assumption, I classify these traffic coming from three different mobile hosts behind a NAT device.

Detecting the NAT devices and their traffic based on the Internet browser information in the HTTP user agent strings has also some limitations as the following:

- When there are several computers behind a NAT device with the same OS and the same browser (e.g. a lab network where all of the computers have the same OS and the same browser, because they are centrally controlled), this technique could not classify them correctly. Because, it could not find any evidence for different TTL values, OSs, and browsers.

- When one host uses a specific version of a web browser and later it uses another version of the same browser, L4 technique could detect these as NAT devices even though they are not. This may happen when the user updates his/her web browser.

- There are several examples of HTTP user agent strings where they do not have any information about the OS and the web browser of the client. Under such conditions, L4 technique could not work accurately.

In summary, if I take all combinations into consideration: There are three Windows NT 5.1 OSs with the same TTL value but two different browser families (types) and different versions, Table 9. Then, there are two Windows NT 6.0 with the same TTL value but different browser families. This can mean two different users or one user with two different browsers. Finally, the packet with Intel Mac OS X 10.5 OS might be a different host (user) or might be the same host (user) with dual OSs. In this case, I use the same assumptions as Maier et al. that two different versions of Windows OSs (NT 5.1 and NT 6.0) would not be installed on the same machine. Moreover, two different versions of the same browser would not be installed on the same machine, either. Therefore, I detect them as three hosts by analyzing these packets. Given that above assumptions, L4 classifies that there are three hosts in Table 10 and two hosts in Table 11.

3.5. Proposed Machine Learning Approach

As discussed earlier, I only used features that are based on flow statistics without using payload information, IP addresses and port numbers. My aim here is to generate automatic signatures via ML techniques without using any biased features. To this end, I have employed two learning techniques: a decision tree system, namely C4.5, and a probabilistic system, namely Naive Bayes. The following summarizes the learning techniques employed.

3.5.1. C4.5

C4.5 is a decision tree based classification algorithm developed by Ross Quinlan that is an extension of the basic ID3 algorithm [9]. C4.5 is designed to address the issues that are not

considered in ID3 such as choosing the appropriate attribute (based on information gain), trying to reduce error pruning, and handling varieties of attributes types (continuous, number, string).

A decision tree is a hierarchical data structure for implementing a divide-and-conquer strategy. C4.5 is an efficient non-parametric method that can be used both for classification and regression. In non-parametric models, C4.5 constructs decision trees from a set of training data applying the concept of information entropy. The training data is a set, *S*, such that each input of the set is an instance of already classified samples. Each sample in the set is a vector where each element in the vector represents an attribute of the sample. The training data is added to a vector where each input in the vector represents a class that each sample belongs to. C4.5 can split the data into smaller subsets using the fact that each attribute of the data can be used to make a decision. Therefore, the attribute with highest information gain is used to make the decision of the split. As a result, the input space is divided into local regions defined by a distance metric. In a decision tree, the local region is identified in a sequence of recursive splits in small number of steps. A decision tree is composed of internal decision nodes and terminal leaves. Each node, *m*, implements a test function $f_m(x)$ with discrete outcomes labeling the branches. This process starts at the root and is repeated until a leaf node is hit. The value of a leaf node constitutes the output. In the case of a decision tree for classification, the goodness of a split is quantified by an impurity measure. A split is pure if for all branches, for all instances choosing a branch belongs to the same class after the split. One possible function to measure impurity is entropy, Eq. (1) [21].

$$I_m = -\sum_{j=1}^{n} p_m^i \log_2 p_m^i \qquad (1)$$

If the split is not pure, then the instances should be split to decrease impurity, and there are multiple possible attributes on which a split can be done. Indeed, this is locally optimal; hence there is no guarantee of finding the smallest decision tree. In this case, the total impurity after the split can be measured by Eq. (2) [21].In other words, when a tree is constructed, at each step the split that results in the largest decrease in impurity is chosen. This is the difference between the impurity of data reaching node m, Eq. (1), and the total entropy of data reaching its branches after the split, Eq. (2).Figure 10 presents the construction of a classification tree. A more detailed explanation of C4.5 algorithm can be found in [21].

$$\boldsymbol{I}_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{k} p_{mj}^{i} \log p_{mj}^{i} (2)$$

```
GenerateTree(X)
    If NodeEntropy(X)< θ_I /* equation 9.3 */
      Create leaf labelled by majority class in X
      Return
    i ← SplitAttribute(X)
    For each branch of x_i
      Find X_i falling in branch
      GenerateTree(X_i)

SplitAttribute(X)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
          Split X into X_1,...,X_n by x_i
          e ← SplitEntropy(X_1,...,X_n) /* equation 9.8 */
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
          For all possible splits
              Split X into X_1, X_2 on x_i
              e←SplitEntropy(X_1, X_2)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Figure 10 Construction of a classification tree [21]

3.5.2. Naive Bayes

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. Depending on the precise nature of the probability model, Naive Bayes classifiers can be trained efficiently in a supervised learning approach. In many practical applications, parameter estimation for Naive Bayes models uses the method of maximum likelihood [14]. A simple Naive Bayes probabilistic model can be expressed as Eq. (3) in the following:

$$P(C|F_1, F_2, \ldots, F_n) = \frac{1}{Z} P(C) \prod_{i=1}^{n} P(F_i|C), \quad (3)$$

where $P(C|F_1, F_2, \dots, F_n)$ is the probabilistic model over dependent class variable *C* with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n; *Z* is a scaling factor dependent only on $F_1, F_2, \dots, F_n$, i.e., a constant if the value of the feature variables are known. A Naive Bayes classifier combines the probabilistic model with a decision rule that aims to maximize a posterior, thus the classifier can be defined using Eq. (4) as follows:

$$Classify(f_1, f_2, \dots, f_n) = argmax_c P(C = c) \prod_{i=1}^{n} P(F_i = f_i | C = c) \quad (4)$$

An advantage of the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

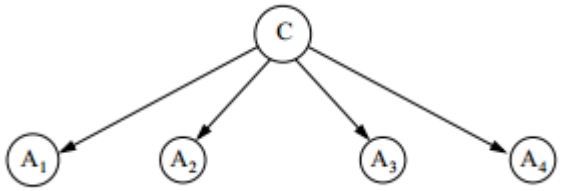


Figure 11 An example of Naive Bayes [14]

Figure 11 shows an example of Naive Bayes. Naive Bayes is the simplest form of Bayesian network. All attributes are independent given the value of class variable. This is called conditional independence. The conditional independence assumption is not often true in the real world problems.

The authors of [12] aimed to show that the Naive Bayes classifier might have been successful to choose who would reply to the mailing for the 1998 KDD Data cup. They evaluated their tests and explained time and space complexities of Naive Bayes by drawing graphs. Time complexity for learning a Naive Bayes classifier is *O(Np)*, where *N* is the number of training examples and *p* is the number of features. Space complexity for Naive Bayes algorithm is *O(pqr)*, where *p* is the

number of features, q is values for each features, and r is alternative values for the class. A more detailed explanation of Naive Bayes algorithm can be found in [14].

CHAPTER 4 EXPERIMENTS AND RESULTS

In this research, I study and evaluate the aforementioned two approaches, namely a ML based approach and a passive fingerprinting based approach. For my proposed ML based approach, I employ the classification models, C4.5 and Naïve Bayes learning techniques introduced in the previous chapter, section 3.5, via an open source tool called Weka [17]. As for the passive fingerprinting approach, I re-engineer and employ the algorithm introduced by Maier et al. [2]. For both approaches, I have used exactly the same data sets including both the encrypted and the non-encrypted traffic as well as aiming to understand the behavior of NAT both with and without the payload information. Moreover, for the proposed approach, I have employed flow (packet header) only features to detect the presence of NAT devices. The following describes the data sets and experiments performed in this research.

In these experiments, I measure the performance of all the techniques employed using two metrics, namely Detection Rate (DR) and False Positive Rate (FPR). DR reflects the number of NAT traffic flows correctly classified. It is calculated using Eq. (4):

$$DR = TP / (TP+FN) \quad (4)$$

where False Negative (FN) reflects the number of NAT flows incorrectly classified as OTHER flows, i.e. non-NAT flows. On the other hand, FPR reflects the number of OTHER flows incorrectly classified as NAT flows using Eq. (5):

$$FPR = FP/ (FP+TN) \quad (5)$$

Naturally, a high DR and a low FPR are the desirable outcomes.

4.1 The Performance of Passive Fingerprinting Approach

I applied all the passive fingerprinting classifiers, namely L1, L2, L3 and L4 techniques, to my data sets to identify the presence of NAT behavior. These results are discussed in the following.

4.1.1 L1

L1 classification technique aims to detect the presence of NAT behavior based only on the TTL values present in the traffic traces. As can be seen in Table 12, in this case, the DR is 0% and FPR is 100% for both of the data sets. The reason is that L1 requires the prior knowledge about the location of the capturing point (where the data is collected). Since each router that forwards a packet is assumed to decrease the TTL by at least one, the TTL is effectively a hop count limit on how far a datagram can propagate through the Internet. However, I do not have any prior knowledge about the location of the capturing point in the Nims-NAT and Partner-NAT networks. Even if I had that knowledge, my closer look to these data sets showed that the NAT devices do not decrease the TTL values on outgoing packets on these networks.

Figure 12 shows the different TTL value ranges for the Nims-NAT data set. As can be seen from these figures, they do not fall in the range, ttl_{init}-1 and ttl_{init}-3, as given by Maier et al. [2].

Table 12 Test Results on the Nims-NAT and the Partner-NAT data sets by using the L1 classifier

	Class-NAT		**Class-OTHER**	
Data sets	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	0%	100%	0%	100%
Partner-NAT	0%	100%	0%	100%

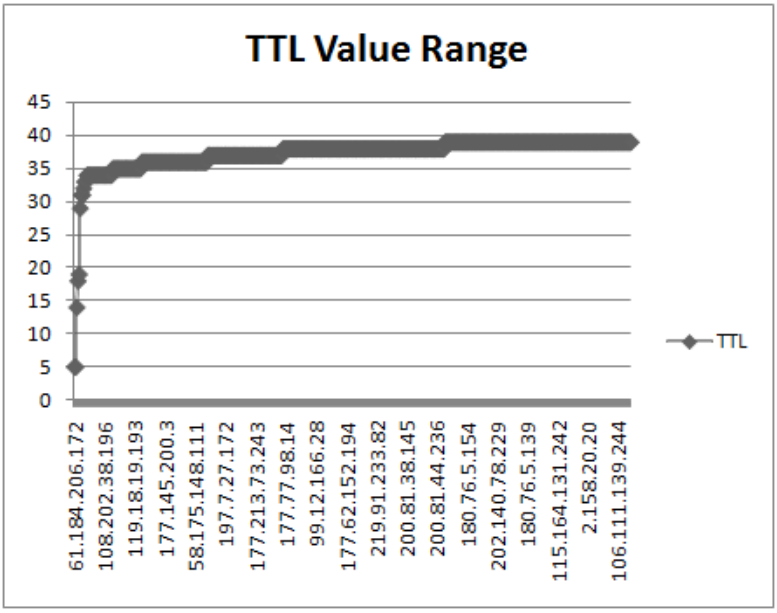


Figure 12 TTL range for Microsoft Windows versions (MS Windows 95/98/98 SE etc.) in the Nims-NAT data set

4.1.2 L2

L2 classification technique aims to detect the presence of NAT traffic based on the TTL range and the distinct TTL values per IP address. Table 13 presents the DR and FPR results for this classifier on our data sets. In this case, the DR of the NAT traffic in the Nims-NAT data set is 100% because the real NAT devices in this data set has more than one TTL value (64 and 128), so all distinct instances belonging to this IP address are detected. In the Nims-NAT data set, there are 12,442 NAT traffic flows (out of 177,493 flows in total) and all are identified (detected) correctly. However the DR for the Partner-NAT data set is 0%. Since all the OSs that belong to the NAT IP addresses in the Partner-NAT data set are Windows XP, there is no NAT device IP address with more than one TTL value in this data set. Moreover, for both the Nims-NAT and Partner-NAT data sets, there are some flows that belong to the hosts that have both Windows and Linux OSs. Thus, these hosts IP addresses have more than one distinct TTL value observed in the data sets. Therefore this results in the L2 technique to identify them as NAT devices and causes the FPR to be 2.7% for the Partner-NAT data set.

Table 13 Test Results on the Nims-NAT and the Partner-NAT data sets by using the L2 classifier

	Class-NAT		Class-OTHER	
Data sets	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	100%	0.16%	99.8%	0%
Partner-NAT	0%	2.7%	97.23%	100%

4.1.3 L3

L3 classification technique aims to detect the presence of NAT behavior based on the TTL Range, the distinct TTL values per IP address, and the different OS information in the HTTP user agent strings per IP address. After parsing HTTP user agent strings in Nims-NAT data set, I obtain OSes in different categories as shown in Table 14 below.

Table 15 shows the DR and FPR for this classifier on our data sets. In this case the FPR of the L3 classifier on the Nims-NAT data set is more than the FPR of the L2 (Table 13) classifier on the same data set even though L3 classifier employs payload inspection, i.e. HTTP user agent string. The reason is that in the Nims-NAT data set, I have some instances that belong to the hosts with two OSs, but both of those OSs are two versions of the same OS, e.g. Windows XP and Windows 7, so L3 classifier automatically detects them as NAT devices, which is obviously not correct. As for the Partner-NAT data set, the DR of NAT flows is still 0% because L3 classifier cannot detect the NAT flows coming from hosts that use the same version of the same OS behind the NAT device. In the Partner-NAT data set, the OS of all the hosts that generate the traffic behind a NAT device is Windows XP.

Table 14 OSs in Nims-NAT data set

Windows	Linux	Macintosh	IOS	Android	BlackBerry
Windows 98	Linux i686	Mac OS X 10.5	CPU iPhone OS 5_0_1	Android 4.0.4	BlackBerry 9300
Windows 3.1	Linux x86_64	Mac OS X 10.6	CPU iPhone OS 6_0	Android 4.1.1	BlackBerry 9780
Windows NT		Mac OS X 10.7	CPU iPhone OS 6_0_1	Android 4.1.2	BlackBerry 9790
Windows XP		Mac OS X 10_6_3	CPU iPhone OS 4_0	Android 1.6	BlackBerry 9800
Windows NT 5.0		Mac OS X 10_6_6	CPU iPhone OS 4_1	Android 2.1	BlackBerry 9900
Windows NT **5.1**		Mac OS X 10_5_8	CPU iPhone OS 4_2_1	Android 2.2.1	
Windows NT 5.2		Mac OS X 10_6_8	CPU iPhone OS 4_3	Android 2.2.2	
Windows NT 6.**0**		Mac OS X 10_7_0	CPU iPhone OS 4_3_3	Android 2.2	
Windows NT 6.1		Mac OS X 10.8	CPU iPhone OS 4_3_5	Android 2.3.3	
Windows NT 6.2		Mac OS X 10_7_1	CPU iPhone OS 5_1_1	Android 2.3.4	
		Mac OS X 10_7_2	CPU iPhone OS 3_1_3	Android 2.3.5	
		Mac OS X 10_7_3	CPU iPhone OS 4_3_1	Android 2.3.6	
		Mac OS X 10_7_4	CPU OS 5_0	Android 2.3.7	
		Mac OS X 10_7_5	CPU OS 5_0_1	Android 3.1	
		Mac OS X 10_8	CPU OS 5_1	Android 3.2	
		Mac OS X 10_8_0	CPU OS 5_1_1	Android 4.0.3	
		Mac OS X 10_8_1	CPU OS 6_0	Android 4.0.4	
		Mac OS X 10_8_2	CPU OS 6_0_1		
		Mac OS X 10.4	CPU OS 3_2		
		Mac OS X 10.5	CPU OS 3_2_2		
		Mac OS X 10_4_11	CPU OS 4_2_1		
		Mac OS X 10_5_3	CPU OS 4_3		
		Mac OS X 10_5_5	CPU OS 4_3_3		
		Mac OS X 10_5_6			
		Mac OS X 10_5_7			
		Mac OS X 10_6			
		Mac OS X 10_5_4			
		Mac OS X 10.8.1			

Table 15 Test Results on the Nims-NAT and the Partner-NAT data sets by using the L3 classifier

	Class-NAT		**Class-OTHER**	
Data sets	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	100%	0.93%	99.6%	0%
Partner-NAT	0	2.7%	97.3%	100%

4.1.4 L4

L4 classification technique aims to detect the presence of NAT behavior based on the TTL Range, the distinct TTL values per IP address, the different OS and the browser information in the HTTP user agent strings per IP address. As shown in Table 17, the FPR of L4 classifier on the Nims-NAT data set is very high (6%). The reason is that there is a DHCP server that assigns the IP addresses randomly to the mobile devices (e.g. smart phones and laptops) on this network. These devices have different versions of the same web browser and might end up using the same IP address during different times of the day. Then, the L4 classifier based on this browser information (from the HTTP user agent strings) categorizes them as NAT devices even though they are not and hence the high FPR. On the other hand, L4 classifier works much better (DR: 100%, FPR: 3%) on the Partner-NAT data set than the L1, L2 and L3 classifiers. Table 17 shows different browser families and browser versions that I observed in the Nims-NAT data set.

Table 16 Test Results on the Nims-NAT and the Partner-NAT data sets by using the L4 classifier

	Class-NAT		**Class-OTHER**	
Data sets	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	100%	6%	93.9%	0%
Partner-NAT	100%	2.7%	97.2%	0%

4.2 The Performance of the Proposed Approach

As can be seen in the previous section, each classifier used for passive fingerprinting approach to identify NAT behavior in the monitored data has some drawbacks on one data set or the other. This not only shows that detecting the NAT behavior in the monitored (analyzed) traffic is challenging, but also it shows that there are different NAT behaviors depending on the organization (Nims versus Partner networks). Moreover, each organization's data can be reflected differently depending on at which level it is analyzed, i.e. network layer (network traffic flows) versus application layer (HTTP user agent strings). Based on our evaluations presented above, the L2 classifier among the passive fingerprinting techniques was the best for the Nims-NAT data set, while the L4 classifier was the best for the Partner-NAT data set.

Table 17 Browsers and versions in Nims-NAT data set

Internet Explorer	Firefox	Chrome	Safari	Opera
MSIE 9.0	Firefox/12.0	Chrome/22.0.1229.94	Mobile Safari/533.1	Opera 12.02
MSIE 8.0	Firefox/14.0.1	Chrome/19.0.1084.52	Mobile Safari/8536.25	Opera 8.01
MSIE 4.01	Firefox/16.0	Chrome/13.0.782.112	Mobile Safari/7534.48.3	Opera 11.64
MSIE 5.0	Firefox/13.0.1	Chrome/18.0.1025.166	Safari/531.21.10	Opera 12.0
MSIE 5.01	Firefox/15.0.1	Chrome/17.0.963.84	Safari/534.56.5	Opera 7.60
MSIE 5.5	Firefox/17.0	Chrome/22.0.1229.96	Safari/536.25	Opera 9.52
MSIE 6.0	Firefox/13.0	Chrome/21.0.1180.89	Safari/536.26.14	Opera 7.64
MSIE 7.0	Firefox/14.0	Chrome/17.0.1000.0	Safari/533.21.1	Opera 11.10
MSIE 10.0	Firefox/8.0	Chrome/22.0.1229.79	Safari/533.19.4	Opera 11.61
	Firefox/4.0	Chrome/19.0.1036.7	Safari/533.22.3	Opera 8.50
	Firefox/10.0.10	Chrome/23.0.1271.60	Mobile Safari/6533.18.5	Opera 11.62
	Firefox/10.0.2	Chrome/17.0.963.56		Opera 10.63
	Firefox/10.0.3	Chrome/12.0.742.5		Opera 12.10
	Firefox/11.0	Chrome/24.0.1312.2		Opera 11.51
	Firefox/13.0	Chrome/23.0.1271.64		Opera 11.52
	Firefox/14.0	Chrome/15.0.861.0		Opera 10.0
	Firefox/3.6.10	Chrome/17.0.1000.0		Opera 12.01
	Firefox/3.0.5			
	Firefox/3.0.7			
	Firefox/3.6.11			
	Firefox/3.6			
	Firefox/16.0.1			
	Firefox/16.0			
	Firefox/4.0.1			
	Firefox/7.0.1			
	Firefox/6.0.2			
	Firefox/3.6.16			
	Firefox/3.5.3			
	Firefox/7.0			
	Firefox/5.0			
	Firefox/9.0			
	Firefox/8.0.1			
	Firefox/9.0.1			
	Firefox/7.0			
	Firefox/2.0.0.20			
	Firefox/3.0.10			
	Firefox/3.6.18			
	Firefox/3.6.11			
	Firefox/3.6.15			
	Firefox/3.6.28			
	Firefox/3.6.7			
	Firefox/3.6.16			
	Firefox/3.6.17			
	Firefox/3.6.4			

On the other hand, when I employed my proposed approach based on the ML classifiers, the performance results of the C4.5 learning technique seems to be well generalized from one data set to the other. This is achieved without using any HTTP user agent strings, i.e. without any application level information, as well as without any aforementioned a priori information. Table 4 shows the Netmate features and Table 5 shows the Tranalyzer features used to represent the traffic to the ML algorithms. In this case, I trained both the C4.5 and the Naïve Bayes learning algorithms using a portion of the network flow data sets from Nims-NAT and Partner-NAT data sets. To this end, I have employed both a balanced and an unbalanced training data of network traffic flows from each data set. I used the Uniform Distribution Filter[1] from Weka to randomly select the training instances to form the training data set for training both of the ML classifiers. Once the classifiers are trained on this data set, I used all the unseen data that is not included in my training phase for testing purposes.

4.2.1 Performance of the Proposed Approach Using Netmate as the Flow Generator

As I mentioned in the previous sections, I used two different flow generator tools to identify NAT devices in the given data sets. As discussed earlier, Section 3.3.1, Netmate is one of the flow generator tools that I employed in this research. Table 2 and Table 3 show the number of flows used for training and testing for each data set. It should be noted here that these are the same data sets used for the passive fingerprint classifiers.

4.2.1.1 Binary (Two-way) Classification

As it is explained in Section 3.2.1 that I first labeled the data sets as NAT and OTHER. Then I do tests using my proposed system which is based on ML techniques. Table 18 shows the training results and Table 19 presents the performance of my proposed approach, where I

[1]Uniform Distribution filter is set in Weka by following this path: weka.filters.supervised.instances.spreadsubsample. This filter produces random subsamples of a data set by using the options options; -S, -M, -W and -X. -M is the maximum class distribution spread. If it is chosen as 1.0, the class values are chosen equally in the manner of uniform distribution.

compared two different ML algorithms for detecting NAT behaviors in Nims-NAT and Partner-NAT data sets. Figure 13 also shows the training results by using the visualization tool, Circos [22].

Table 18 Training Results by using the proposed approach with the Netmate features

		Class-NAT		Class-OTHER	
		DR	**FPR**	**DR**	**FPR**
Nims-NAT Partner-NAT (merged)	**C4.5 based classifier**	99.3%	1.3%	98.7%	0.7%
	Naive Bayes based classifier	17.6 %	5.6%	94.4%	82.4%

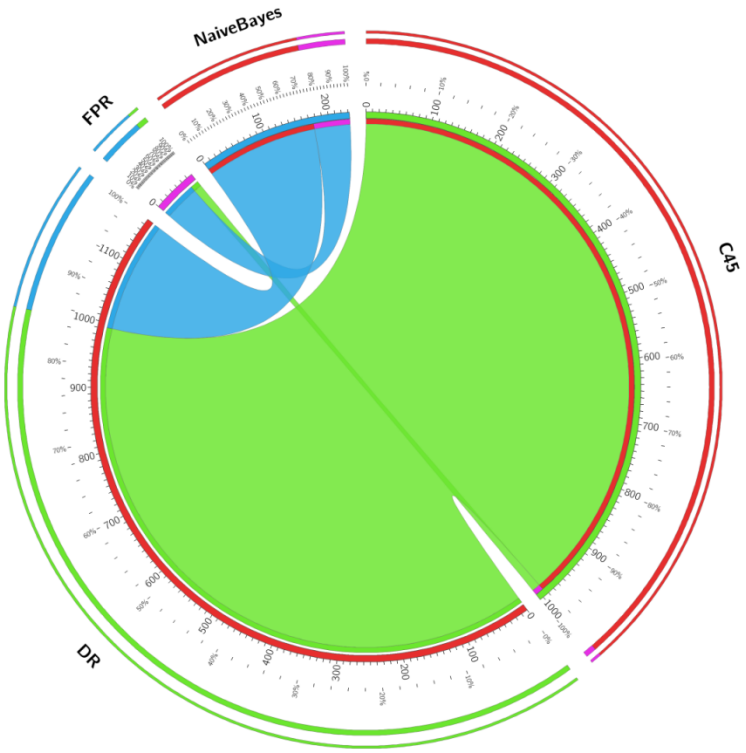


Figure 13 Image shows that the DR and the FPR values as a result of C4.5 and Naive Bayes algorithms for the NAT class on the training data set

My proposed system using C4.5 classifier outperforms the L1 classifier of the passive fingerprinting approach on both data sets in terms of DR and FPR. My proposed system only

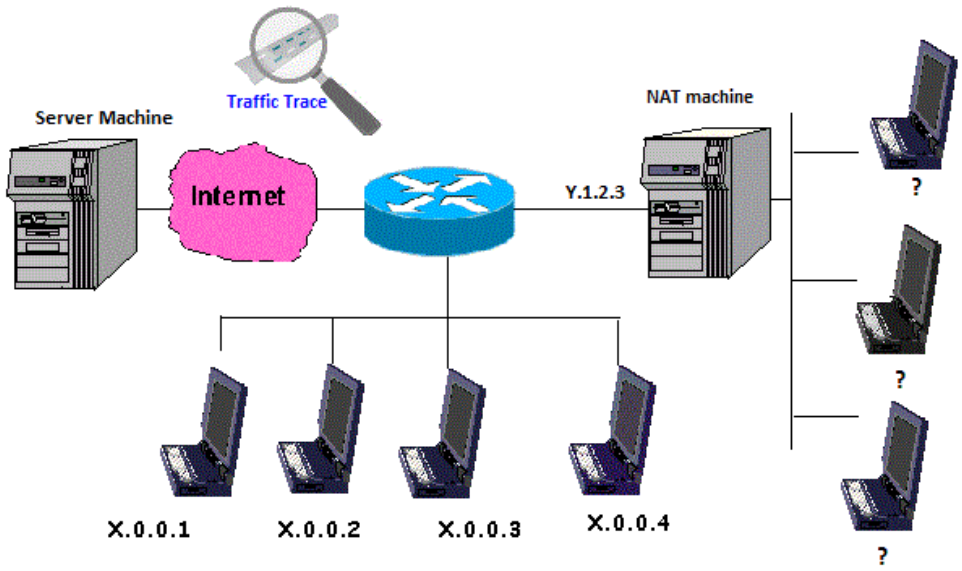
employs network flow based information. Moreover, it also outperforms the L2 and L3 classifier on the Partner data set and performs as good as the L4 classifier on both data sets even though L3 and L4 classifiers employ both the network flow and the HTTP user agent strings information.

Table 19 Testing Results by using the proposed approach with the Netmate features

		Class-NAT		**Class-OTHER**	
		DR	**FPR**	**DR**	**FPR**
Unseen Nims-NAT data set	**C4.5 based classifier**	98.7%	3.7%	96.3%	1.3%
	Naive Bayes based classifier	15%	13%	98%	98%
Unseen Partner-NAT data set	**C4.5 based classifier**	98%	2.4%	97.6%	2%
	Naive Bayes based classifier	34%	10%	89%	66%

C4.5 learning technique based classifier has the property to choose the most appropriate features from the feature set given to it. This property enables me to learn which features of the network flow traffic have contributed to this high performance. Once I analyzed the solution decision tree generated by the C4.5 algorithm, I was able to identify the most helpful features for the classifier to detect different NAT behaviors existing in the network traffic, Figure 14.

These features seem to work for both of the data sets employed in this work. Even though these are features have the highest weights in the solution tree, my system is based on all 41 features of Netmate (Table 4). This also indicates the challenges and different NAT behaviors present in the data sets.



Features
The average number of bytes in a sub flow in the forward direction (sflow_fbytes)
Total bytes in the backward direction (total_bvolume)
The mean size of packets sent in the forward direction (mean_fpktl)
The maximum amaount of time that the flow was active before going idle (max_active)
The size of the smallest packet sent in the forward direction (min_fpktl)
The size of the largest packet sent in the backward direction (max_bpktl)
The standard deviation from the mean of the oacket sent in the backward direction (std_bpktl)

Figure 14 The most important Netmate features selected by the C4.5 classifier

4.2.1.2 Three-Way Classification

As it is also discussed in Section 3.2.1. I only have ground truth regarding encrypted NAT traffic in the Nims-NAT data set. In this case, L1, L2, L3 and L4 techniques are not equipped to distinguish between encrypted an unencrypted NAT flows, so I only evaluate my proposed ML base approach using three way classification (unencrypted-NAT vs. encrypted-NAT vs. Others) on these traffic flows. Table 20shows the training performance and Table 21shows the testing performance of the proposed approach. Figure 15 and Figure 16show the training results in terms of DR and FPR metrics.

Table 20 Training Results by using the proposed approach with the Netmate features

		Encrypted-NAT		Unencrypted-NAT		OTHER	
		DR	**FPR**	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	**C4.5 based classifier**	99.4%	0.3%	99.4%	1.3%	97.5%	0.3%
	Naive Bayes based classifier	35.2%	2.8%	34%	2.8%	93.7%	22.9%

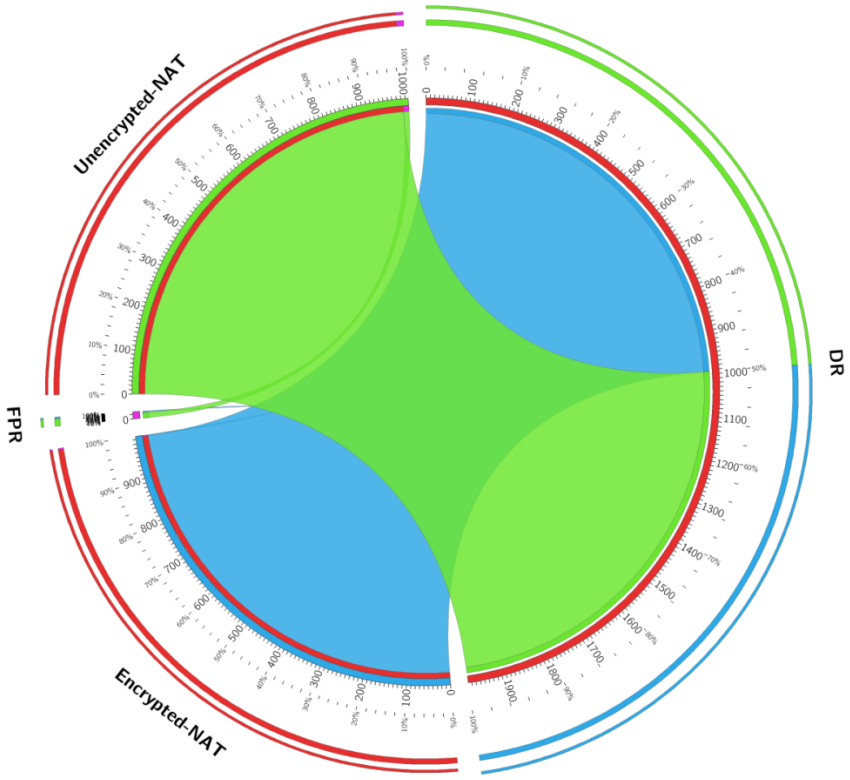


Figure 15 Image shows that the DR and the FPR values as a result of the trained C4.5 algorithm for the encrypted and the unencrypted NATs in the Nims-NAT data set

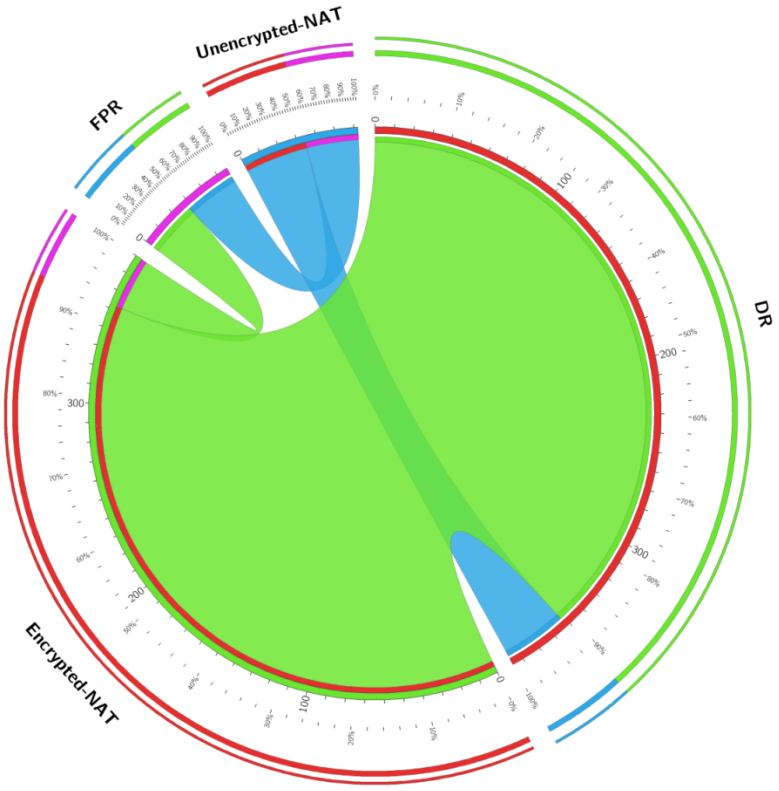


Figure 16 Image shows that the DR and the FPR values as a result of the trained Naive Bayes algorithm for the encrypted and the unencrypted NATs in Nims-NAT data set

Table 21 Testing Results by using the proposed approach with the Netmate features

		Encrypted-NAT		Unencrypted-NAT		OTHER	
		DR	**FPR**	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	**C4.5 based classifier**	98.1%	1.9%	92.9%	7.3%	90.8%	6.2%
	Naive Bayes based classifier	43.4%	5.4%	33.3%	4.1%	90.4%	62.4 %

These results show that the C4.5 based classifier gives very promising performance to identify NAT devices even in the encrypted traffic. Table 21 shows that differentiating encrypted NAT traffic from unencrypted NAT traffic and from other traffic is challenging but the proposed approach using the C4.5 based classifier can do this relatively well (above 90% DR) albeit the FPR also increases relative to binary classification.

4.2.2 Performance of the Proposed Approach Using Tranalyzer as the Flow Generator

I repeated my tests for both Nims-NAT data set and Partner-NAT data set by using Tranalyzer as a different flow generator tool. As it is discussed in *Feature Selection* section, Tranalyzer generates a larger number of features, which are shown in Table 5, than Netmate. Moreover, most of these features are different than the ones Netmate generates. Therefore, my aim here is to test whether these new and more number of features would have any effect on the performance of the proposed approach. Tables 2 and Table 3 show the number of flows used for training and testing for each data set.

4.2.2.1 Binary (Two-Way) Classification

In this case, Table 22 shows the results for the training phase using the same training data set (in terms of network packets employed) as in section 4.2.1.1. Again, I evaluate the performances by testing the model on both the Nims-NAT and the Partner-NAT data sets separately. Table 23presents the results for the testing phase.

Table 22 Training Results by using the proposed approach with the Tranalyzer features

		Class-NAT		**Class-OTHER**	
		DR	**FPR**	**DR**	**FPR**
Nims-NAT Partner-NAT (merged)	**C4.5 based classifier**	99.8%	0.7%	99.3%	0.2%
	Naive Bayes based classifier	67.3%	8.1%	91.9%	32.7%

According to the test results, Table 23 below, again C4.5 based classifier detects both NAT and non-NAT classes better than the Naive Bayes classifier. The DR and the FPR are especially good for the Nims-NAT data set. However, the performance of the proposed approach is not as good as the performance when Netmate features were used on the same data sets, Section 4.2.1.1.

Table 23 Testing Results by using the proposed approach with the Tranalyzer features

		Class-NAT		Class-OTHER	
		DR	**FPR**	**DR**	**FPR**
Unseen Nims-NAT data set	**C4.5 based classifier**	98.7%	1.6%	98.4%	1.3%
	Naive Bayes based classifier	68%	8.1%	91.9%	32%
Unseen Partner-NAT data set	**C4.5 based classifier**	72%	55%	57%	27%
	Naive Bayes based classifier	58%	8%	91%	41%

4.2.2.2 Three-Way Classification

In this case, Table 24presents the training results and Table 25 presents the testing results of my proposed approach, where I compared two different ML algorithms for detecting encrypted-NAT behaviors in Nims-NAT data set. Figure 17 and Figure 18show the training results in terms of DR and FPR metrics.

Table 24 Training Results for the three-way classification by using the proposed approach with the Tranalyzer features

		Encrypted-NAT		Unencrypted-NAT		OTHER	
		DR	**FPR**	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	**C4.5 based classifier**	96%	3%	93.5%	1.6%	99.4%	1.1%
	Naive Bayes based classifier	0	0	96.6%	53.4%	86.9%	4.8%

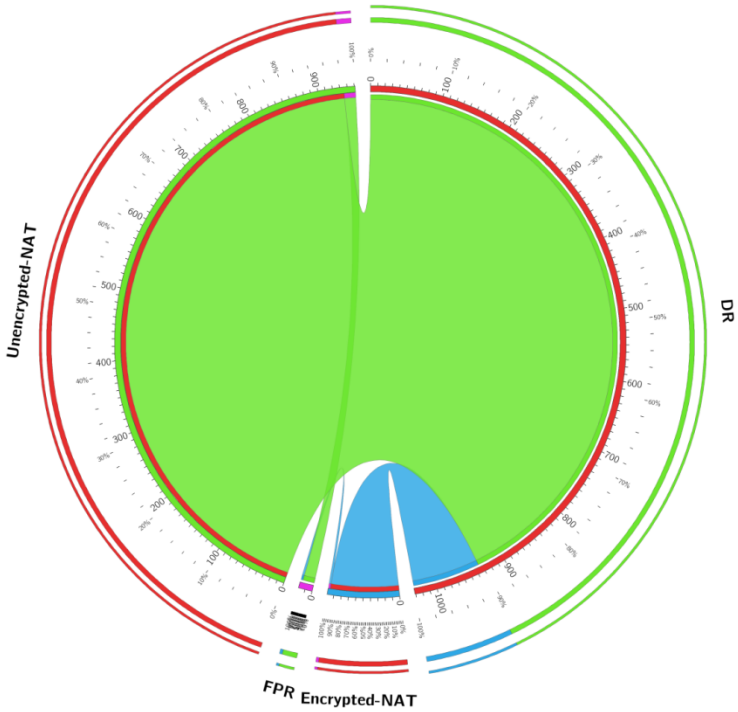


Figure 17 Image shows that the DR and the FPR values as a result of C4.5 based classifier for encrypted and unencrypted NATs in the Nims-NAT training data set

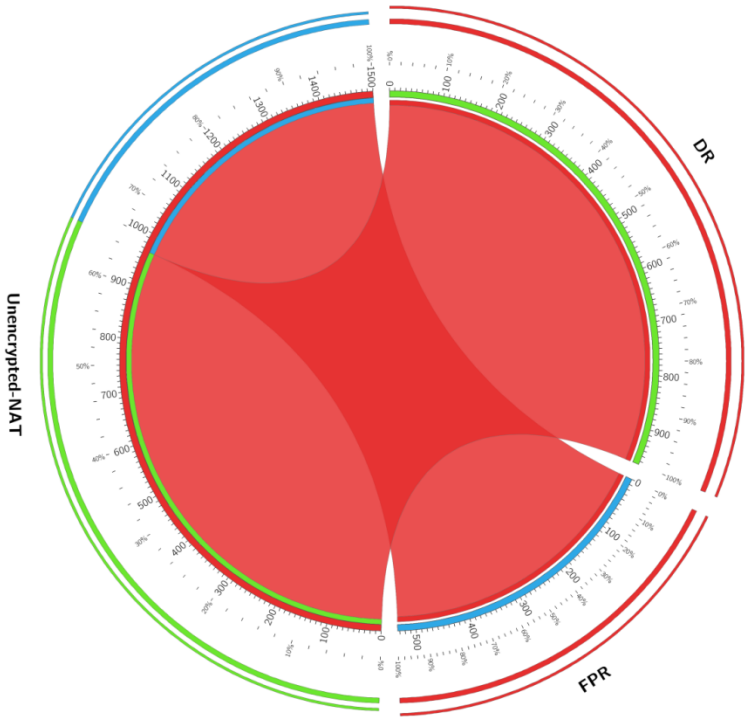


Figure 18 Image shows that the DR and the FPR values as a result of Naive Bayes based classifier for encrypted and unencrypted NATs in the Nims-NAT training data set

Table 25 Testing Results for the three-way classification by using the proposed approach with the Tranalyzer features

		Encrypted-NAT		Unencrypted-NAT		OTHER	
		DR	**FPR**	**DR**	**FPR**	**DR**	**FPR**
Nims-NAT	**C4.5 based classifier**	67.3 %	5.8%	57.8%	6.1%	90.6%	3.8%
	Naive Bayes based classifier	0%	0.1%	93%	7.4%	85.3%	6.8%

The test results are not as good as the ones when Netmate features were used with classifiers. This evaluation shows that the Netmate features are more effective to identify NAT traffic on these data sets than the Tranalyzer features.

These results show that passive fingerprinting classifiers seem to work for certain NAT behaviors better than the others. Moreover, as the NAT behavior gets more unique and challenging, passive approach requires access to the application (payload) information such as HTTP user agent strings to reach a high DR with low FPR. On the other hand, my proposed approach based on machine learning, specifically C4.5 decision tree learning classifier, enables me to achieve a high performance (high DR and low FPR) accuracy without using any application level data and generalizing well to different NAT behaviors present in different data sets.

4.3 Predicting the Number of Hosts Behind Detected NAT Devices

In this case, once I detect a NAT device, using ML based approach, I can predict the number of users behind a NAT using the OS and browser information whenever it is available. Examples for this are shown in Table 26 and Table 27. In these examples, this heuristic predicts the presence of two users behind the NAT device detected by my proposed learning approach using C4.5 classifier. However, there might be more users using the same combinations. This heuristic will not be able to predict the maximum number.

Table 27 shows ten distinct OS, browser family and versions and TTL combinations which belong to the NAT device that I know the IP address of it as a ground truth in Nims-NAT data set and also in unencrypted traffic. According to my heuristic, I can count two different hosts behind that NAT device as I shown in Figure 19 below. There are two types of OSs: Windows 7 which have the TTL values as 125 and Mac OS X which have the TTL values as 61. Thus, TTL and OS combinations are not enough to predict the number of users behind the NAT device. There are different browser families. It is possible that more than one browser can be installed on one host. I have to check them together with their versions. This shows that there are two hosts. I can say that one of them has two OSs installed which are Windows 7 and Mac OS X. In Windows 7, Internet Explorer 9.0, Firefox 15.0.1 and Google Chrome 23 are installed as browsers, whereas in Mac OS X, Safari 6.0 and Google Chrome 23 are installed. In the second computer, Windows 7 is installed as the OS and Internet Explorer 7.0 and Firefox 14.0.1 installed as the browsers.

Table 26 The OS, Browser Family, Browser Version and IP Combinations in non-encrypted Nims-NAT Data set

	IP Address	OS name	OS version	Processor	Browser	TTL
Unencrypted Traffic	129.173.13.94	Windows 7	Windows NT 6.1	32	Internet Explorer 9.0	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Internet Explorer 9.0	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Internet Explorer 7.0	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Firefox 15.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Firefox 15.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Firefox 14.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Google Chrome 23	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Internet Explorer 7.0	125
	129.173.13.94	Mac	Intel Mac OS X 10_7_4	Intel CPU	Safari 6.0	61
	129.173.13.94	Mac	Intel Mac OS X 10_7_4	Intel CPU	Google Chrome 23	61

Indeed, in these examples, I have the information such as the OSs and the browser information, which are available in the user agent strings, in order to be able to predict the number of users behind the NAT device, Table 27. In another example, there are Windows 7 and Mac OS X OSs installed. When I look at the TTL values, there are 125 for Windows and 61 for Mac OS X again. The same IP address, the same OS and the same TTL value do not help me to guess the hosts. Browser families with their versions are needed. After extracting these information, I can say that there are two hosts behind the NAT device just according to the heuristic as it is shown in Figure 20. One of them has Windows 7 with Internet Explorer 9.0, Firefox 15.0.1, Safari 5.1.7 and Google Chrome 23 installed and Mac OS X with Google Chrome 13, Firefox 14.0.1, Safari 5.1.7 installed. The second host has Windows 7 with Firefox 14.0.1 and Google Chrome 22 and Mac OS X with Safari 5.1.7 installed. Table 27 shows all OS and browsers with their versions.

Figure 19 Two hosts are found behind the NAT device in the unencrypted traffic in Nims-NAT data set

According to my heuristic, I can count two hosts in both encrypted and unencrypted traffic in Nims-NAT data set. However, there might be more than two hosts, because it is possible that each OS is installed on a different computer. There might be computers which have the same TTL value, with the same number of hops away from the NAT device, and also have the same

OS installed. In such cases, there might be the same browsers or different browsers. For instance, in Table 27, the first row and the second row might belong to two different computers. In my opinion, it is reasonable because NAT servers locates generally in labs so the computers in some labs might not need to upgrade their OSs. In that kind of situations, the OS which is installed originally would be the same on all computers.

On the other hand, there might be hosts which have both Windows and Mac OS X OSs installed. As it is seen in Table 26 and Table 27, I can extract just two of these OSs behind a NAT device. That's why I can estimate that these two OSs might be on the same machine. However, this can only be done by having some a priori knowledge regarding the network under analysis.

Table 27 The OS, Browser Family, Browser Version and IP Combinations in encrypted Nims-NAT Data set

	IP Address	OS name	OS version	Processor	Browser	TTL
Encrypted Traffic	129.173.13.94	Windows 7	Windows NT 6.1	32	Internet Explorer 9.0	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Internet Explorer 9.0	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Safari 5.1.7	125
	129.173.13.94	Windows 7	Windows NT 6.1	64	Firefox 15.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Firefox 15.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Firefox 14.0.1	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Google Chrome 23	125
	129.173.13.94	Windows 7	Windows NT 6.1	32	Google Chrome 22	125
	129.173.13.94	Mac	Intel Mac OS X 10_7_4	Intel CPU	Google Chrome 13	61
	129.173.13.94	Mac	Intel Mac OS X 10_7_4	Intel CPU	Firefox 14.0.1	61
	129.173.13.94	Mac	Intel Mac OS X 10_7_4	Intel CPU	Safari 5.1.7	61
	129.173.13.94	Mac	Intel Mac OS X 10_6_8	Intel CPU	Safari 5.1.7	61

As another heuristic, I also extracted the processor information from the user agent strings from the HTTP log files, the processor information could help me to identify the different hosts, too. For instance, in the first and second row of Table 27, one of them has 32 bit Windows 7 and the other one has 64 bit Windows 7 OS. In this case, with high confidence, I can say that these are two separate hosts.

Figure 20 Two hosts are found behind the NAT device in the encrypted traffic in Nims-NAT data set

CHAPTER 5 NAT DETECTION SYSTEM TOOL

In this chapter, I introduce the tool I developed based on my proposed approach that is presented and evaluated in the previous chapters. The name of my tool is NAT Detection System, NAT-Detect, which is developed by using Java programming language in Eclipse Integrated Development Environment (IDE) [23].

First of all, a captured traffic file in the format such as .tcpdump, .pcap, or .dmp needs to be uploaded to NAT-Detect. Then, NAT-Detect runs Netmate as the flow generator tool in the background to convert the packets of the uploaded traffic file into flows. Figure 21 shows a screen shot of the main interface of NAT-Detect. Figure 22 shows the possible outputs when Netmate starts running.

After the conversion, by using *OPEN FLOWS as CSV* button, NAT-Detect provides users the flows as a *csv* file. One of the optional functionality provided by NAT-Detect is an interface where an expert can input the ground truth information for re-training purposes (if/when needed). Figure 23 shows a screen shot for this.

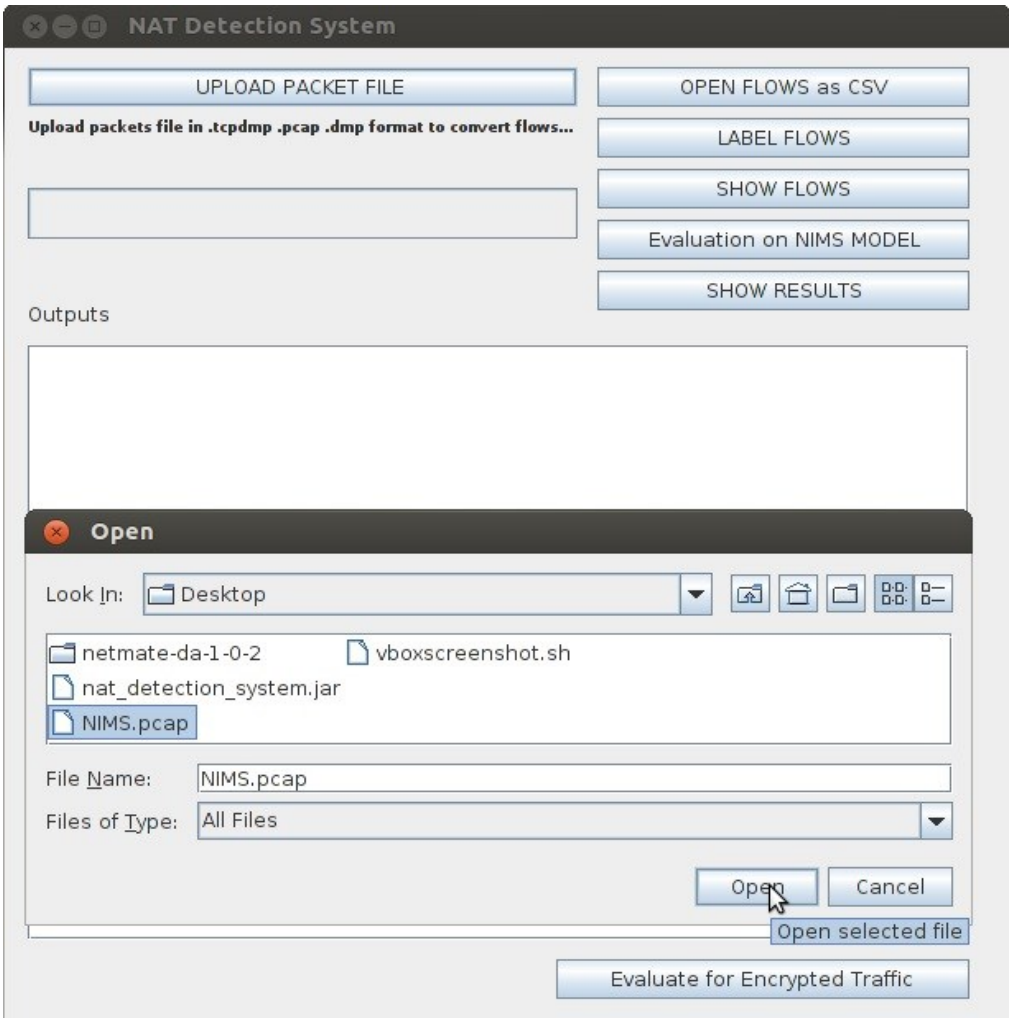


Figure 21 Main Interface of NAT-Detect

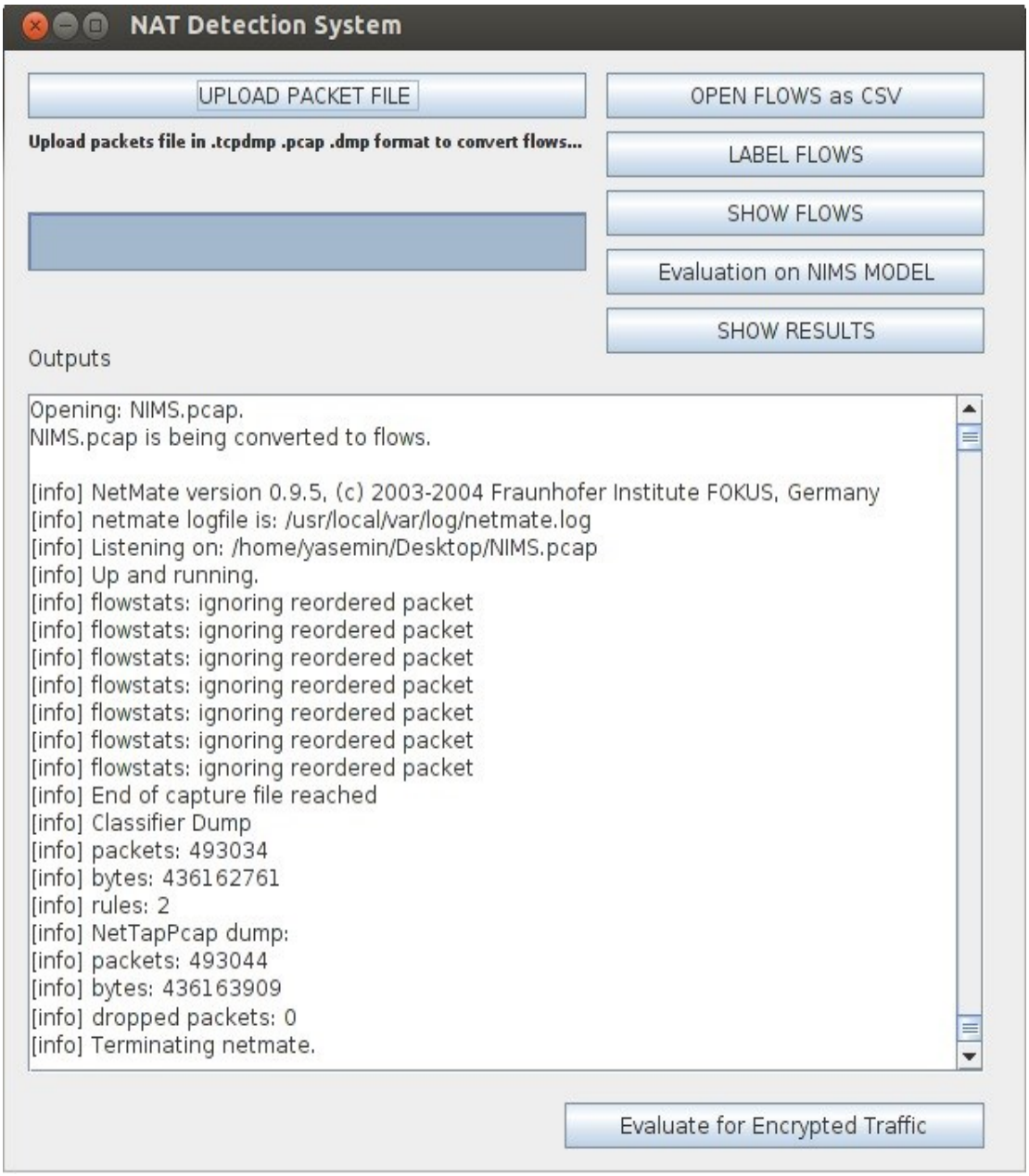


Figure 22 The output of Netmate is shown on the output screen

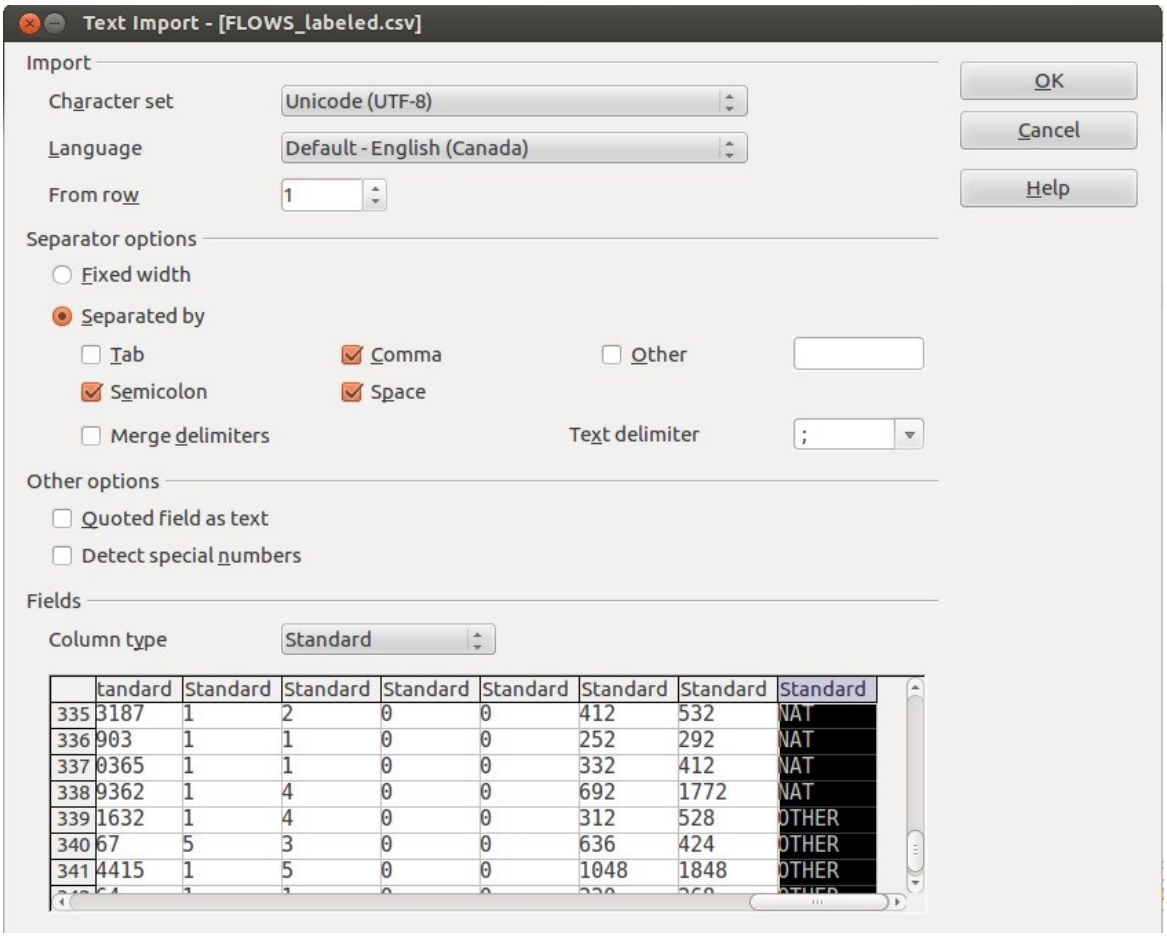


Figure 23 Opening flows in a csv file

If the optional functionality is not used, then by default the C4.5 based classification solution presented in the previous chapters is run on the flows of the uploaded data set. This is the evaluation functionality of NAT-Detect. *Evaluate on NIMS MODEL* button runs this on the flows of the uploaded file. Figure 24 shows an example output for this process and Figure 25 shows the result screen. As a result, the DR, the FPR, the number of instances from each class, and the IP addresses which are classified correctly and incorrectly, and also the number of users predicted behind NAT devices are presented.

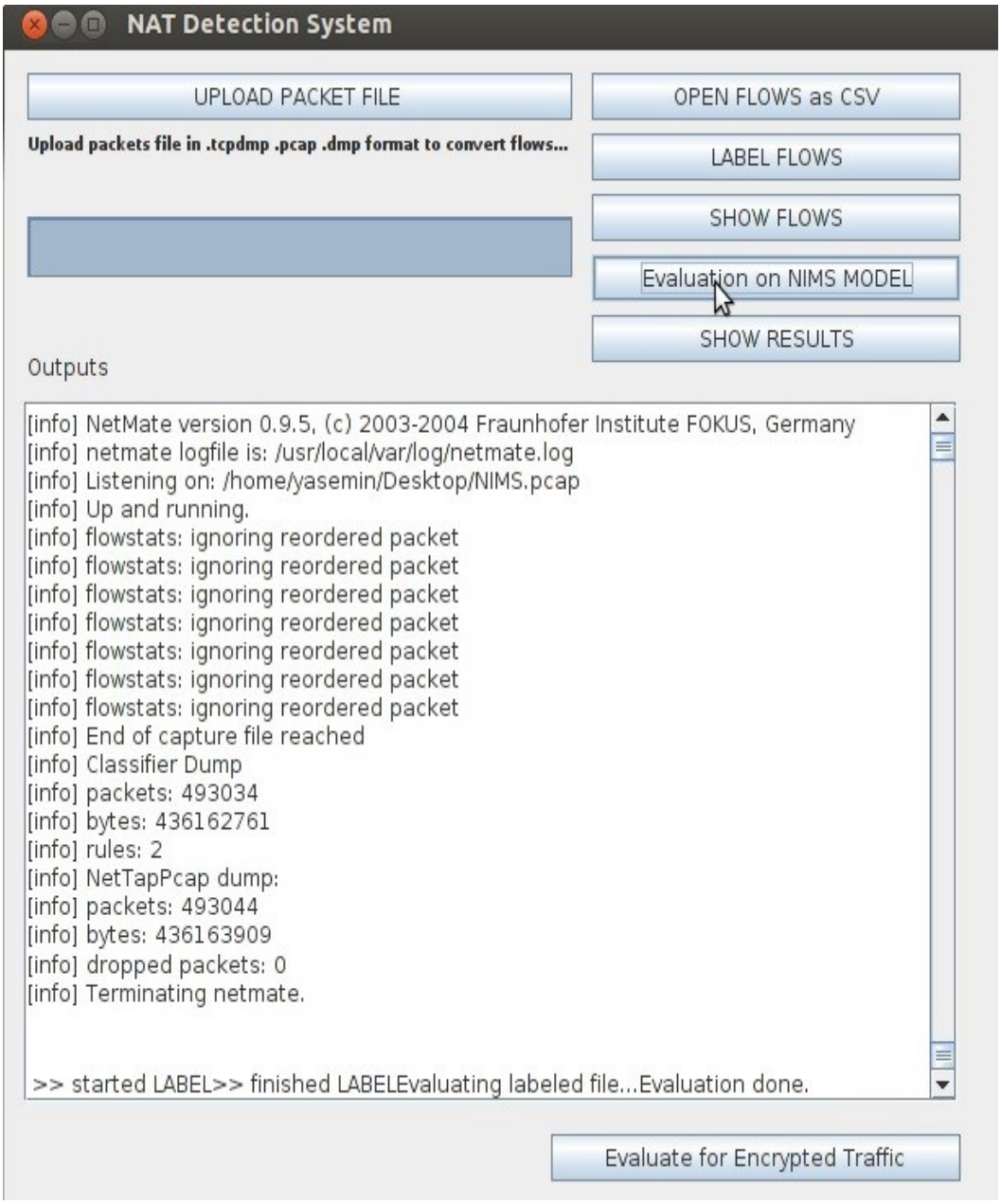


Figure 24 Detecting NAT traffic using the proposed approach

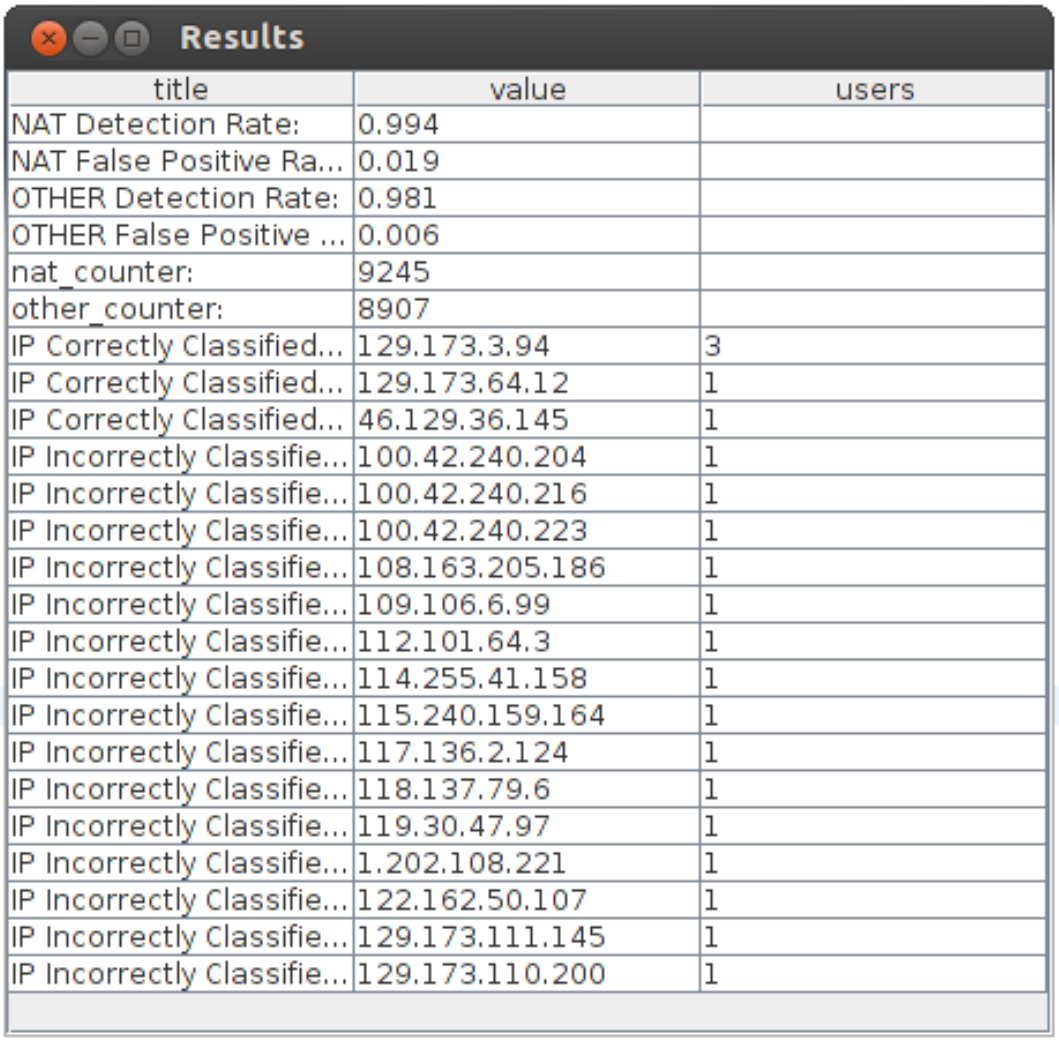


Figure 25 Sample detection result

Moreover, NAT-Detect can also classify the flows, specifically for Encrypted-NAT traffic. "Evaluate for Encrypted Traffic" button in Figure 24 is used for this purpose. Figure 26 shows an example for this purpose.

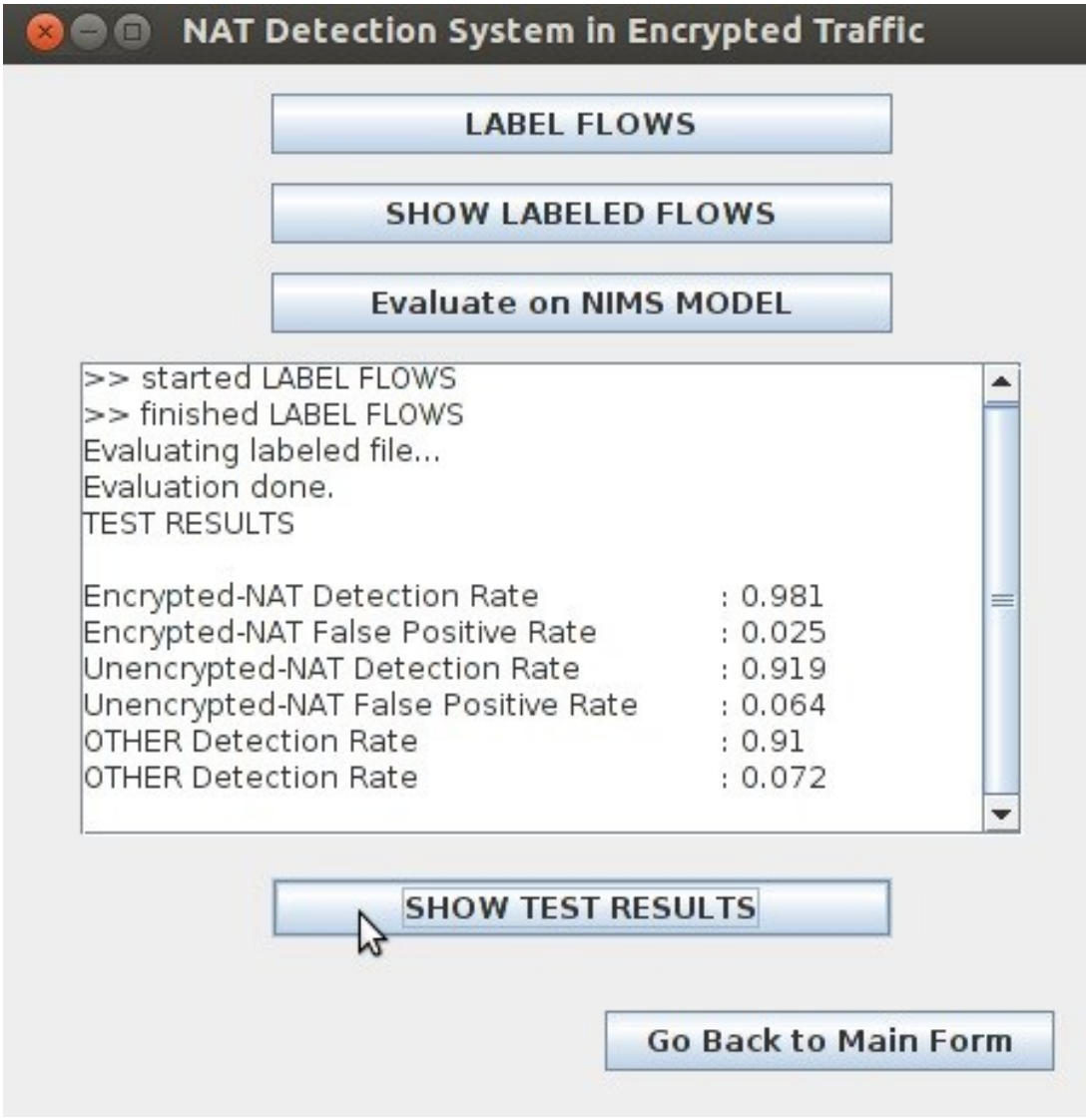


Figure 26 Sample Results for the Encrypted traffic

CHAPTER 6 CONCLUSION

In this research, I explored how far I can push a ML based classification approach to identify NAT devices using only network flows. As a new contribution, identifying NAT behavior by using traffic flows and ML classifiers for forensic analysis was presented in this thesis. To this end, I represented the traffic as network flows to two ML techniques, namely C4.5 and Naive Bayes, without using IP addresses, port numbers and payload (application) information. I evaluated my approach on two different data sets against four different variants of the passive fingerprinting approach [2]. These techniques represent the state-of-the-art techniques employed in the field to detect NAT devices (traffic). Moreover, my proposed system also predicts the number of users behind NAT devices. My results show that the proposed approach using C4.5 learning classifier performs better than the passive fingerprinting techniques on both data sets even though two of the passive fingerprinting techniques employ payload information. This is a very promising result given that payload becomes opaque when encryption is used at the application level. Future work will analyze different traffic data from virtual private networks and cascaded network environments to better understand the potentially different behaviors of NAT devices under such conditions. Moreover, different ML classifiers and flow features may be explored under these new conditions. Finally, how solutions of the C4.5 based classifier can be converted into automatic signatures will be interesting to further study.

BIBLIOGRAPHY

[1] Ishikawa, Y.; Yamai, N.; Okayama, K.; Nakamura, M.; , "An Identification Method of PCs behind NAT Router with Proxy Authentication on HTTP Communication," Applications and the Internet (SAINT), 2011 IEEE/IPSJ 11th International Symposium on , vol., no., pp.445-450, 18-21 July 2011.

[2] G. Maier, F. Schneider, and A. Feldmann. NAT Usage in Residential Broadband Networks. Proceedings of the 12th International Conference on Passive and Active Network Measurement (PAM 2011), Atlanta, Georgia, March 2011.

[3] Murakami, R.; Yamai, N.; Okayama, K.; , "A MAC-address Relaying NAT Router for PC Identification from Outside of a LAN," Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on , vol., no., pp.237-240, 19-23 July 2010.

[4] Li Rui; Zhu Hongliang; Xin Yang; Yang Yixian; Wang Cong; , "Remote NAT Detect Algorithm Based on Support Vector Machine," Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on, vol., no., pp.1-4, 19-20 Dec. 2009.

[5] Phaal, P. Detecting NAT devices using sFlow. http://www.sflow.org/detectNAT/ (last modified: 2009).

[6] Miller, T. Passive OS fingerprinting: Details and techniques. http://www.ouah.org/incosfingerp.htm (last modified: 2005).

[7] R. Beverly. A robust classifier for passive TCP/IP fingerprinting. In Proc. Conference on Passive and Active Measurement (PAM) (2004).

[8] S. M. Bellovin, “A technique for counting natted hosts”, In IMW ’02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, pp. 267–272, New York,NY, USA, 2002, ACM Press.

[9] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[10] VojtechKrmicek , Jan Vykopal , RadekKrejci, Netflow based system for NAT detection, Proceedings of the 5th international student workshop on Emerging networking experiments and technologies, December 01-01, 2009, Rome, Italy.

[11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reute-mann, and I. H. Witten, "The weka data mining software: An update," SIGKDD Explorations, vol. 11, no. 1, 2009.

[12] R. Alshammari, A. N. Zincir-Heywood, Machine learning based encrypted traffic classification: Identifying ssh and skype, IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–8, 2009.

[13] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[14] George H. John and Pat Langley Estimating Continuous Distributions in Bayesian Classifiers. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. pp. 338-345, Morgan Kaufmann, San Mateo, 1995.

[15] Netmate.http://www.ipmeasurement.org/tools/netmate/

[16] IETF.http://www3.ietf.org/proceedings/97apr/97aprfinal/xrtftr70.htm.

[17] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[18] http://josephmlod.wordpress.com/network/mpls-multiprotocol-label-switching/ttl-behavior-of-labeled-packets/

[19] Wireshark. http://www.wireshark.org/

[20] Tranalyzer. http://tranalyzer.com/

[21] E. Alpaydin, Introduction to Machine Learning, MIT Press, 2004

[22] Circos. http://circos.ca

[23] Eclipse. https://www.eclipse.org/