EXPLORING NAT DETECTION AND HOST IDENTIFICATION

by

Lan Zhang

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

at

Dalhousie University
Halifax, Nova Scotia
August 2018

Table of Contents

List of Tables

List of Figures

Abstract

This thesis explores NAT detection and host identification. The NAT detection approach is processed by supervised machine learning algorithms on HTTP attributes. Three classifiers are employed on training datasets labelled by artificial NAT generation method in NAT detection. This research demonstrates that AD Tree performs best in NAT detection and selects five effective attributes for it. AD Tree can detect NAT devices with an accuracy approximately of 100% on five datasets. The impact of difference in sizes of datasets in NAT detection is also observed in this thesis. Host identification is based on TCP timestamp values and system uptime values of TCP packets. This research identifies end hosts behind a detected NAT device using an improved artificial line generation method and an improved line distance calculation method. It also provides a new evaluation method for host identification. These two tasks are combined in this research for forensic analysis in order to analyze cybersecurity incidents that could occur from unknown NAT devices in the incoming traffic to an organization.

List of Abbreviations Used

NAT	Network Address Translation
LAN	Local Area Network
IP	Internet Protocol
HTTP	HyperText Transfer Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
AD Tree	Alternating decision tree
ID3	Iterative Dichotomiser 3
LBS	Location-Based Service
RFC	Request for Comments
TTL	Time to Live
DR	Detection Rate
OS	Operating System
ML	Machine Learning
Weka	Waikato Environment for Knowledge Analysis
TSval	TCP Timestamp Value
UTC	Universal Time Coordinated
GB	Giga Byte
TPR	true positive rate

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Dr. Nur Zincir-Heywood and Dr. Khurram Aziz, for their support, guidance, encouragement and valuable ideas that have helped me complete the research and the thesis. I would also like to thank Dalhousie University, and Faculty of Computer Science for their supports toward this thesis research. I would also like to thank 2Keys Corporation for the opportunities created by the NSERC program. I am particularly grateful for the assistance given by CS help desk on technique support. I would like to offer my special thanks to Yilong Zhu and Yu Bai who have given me confidence and motivation during this summer. Last but not least, I would like to thank my parents and all my friends for their love and support throughout my life and study.

Chapter 1 Introduction

Network address translation (NAT) is a method that allows multiple end hosts of an entire private network to share one Internet-routable IP address of a NAT gateway [1]. Network address information in the IP header field of end hosts' packets is modified by a NAT device into the IP address of the NAT gateway.

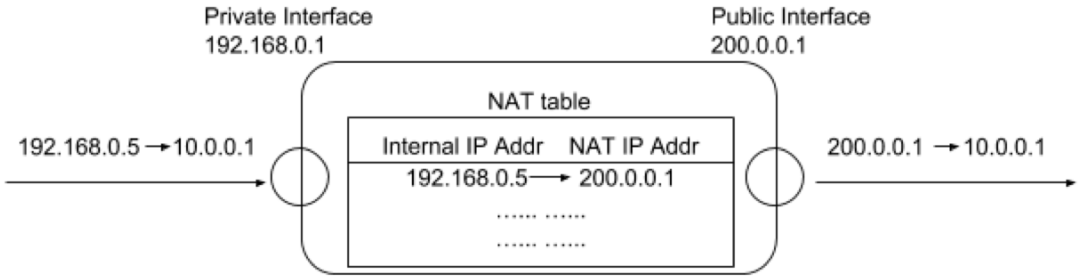

Figure 1.1: NAT device receives packets from external network

The illustration above shows how a NAT device works when a packet is sent from the Internet to a host in the internal network. When one NAT device receives a packet from hosts in the internal network, it alters the source IP address of the packet into its registered IP address before forwarding the packet on. NAT devices create NAT tables to help track all the connections through NAT devices and determine how to modify IP addresses of packets and whom to forward them to. Users from the Internet receive packets with the same source IP address sent from different hosts behind the NAT device and send back packets to that IP address.

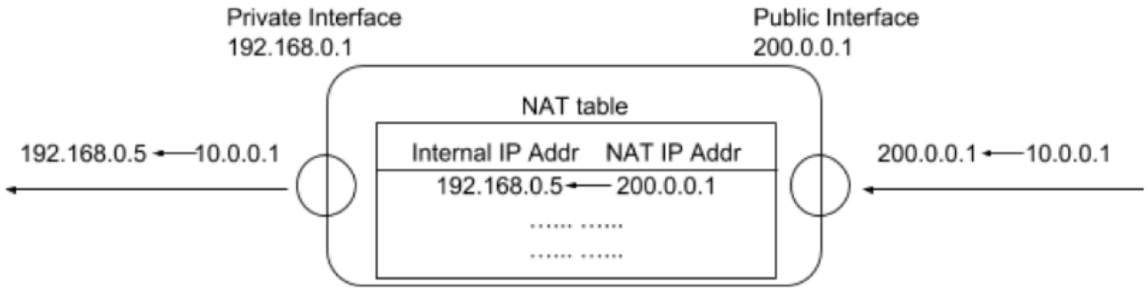

Figure 1.2: NAT device sent back packets to external network

When a NAT device receives a packet sent back from the Internet to hosts behind it, as is shown in the illustration above, it looks up the NAT table to modify the destination IP address (NAT device's IP address) into the IP address of certain host from the internal network and then forwards it to that host.

NAT plays an essential role in managing the IPv4 address exhaustion problem. It can get the most out of a single registered IP address. The size of the network behind a NAT gateway varies, and according to CISCO technology support, typical routing hardware can support at least thousands of NAT translations simultaneously [2]. Hence it is popular for both private home networks with several hosts and local area networks (LAN) of large institutions with multiple hosts.

However, the wide deployment of NAT implementation brings a great security threat to Internet. NAT modifies the IP address information of packets, thus uncareful implementation can have serious consequences for networks. Unauthorized NAT devices which do not meet requirements of careful implementation on the Internet can be easily targeted by attackers. NAT gateways with multiple end hosts can result in more loss than normal hosts if controlled by hackers. The size and topology of end hosts behind a NAT gateway are unknown to users outside the private network. This also brings difficulty in the management of networks containing NAT gateways. Other users do not know whether unauthorized NAT gateways are end hosts or NAT devices.

Even assuming that NAT devices have been distinguished from end hosts, aberrant behavior of a NAT device can be caused by one of the end hosts behind it. Simply blocking that IP address will impact others hosts in the internal network. To manage network traffic precisely, we need to distinguish the hosts behind one NAT gateway from packets with the same source IP address.

Host identification behind a NAT device is certainly necessary for the sake of Internet security. NAT detection is also a requisite step of host identification. These two stages allow us to understand if a host on the Internet is a NAT device or not and identify hosts from its internal network if it is. In Toma´s Kom˘arek's work, the NAT detection problem is addressed by behavior modeling [3]. A supervised machine learning approach is used to build the behavior model of NAT devices from HTTP access logs, or concretely speaking, the Support Vector

Machine (SVM) algorithm is used in his experiment. The difficulty in NAT detection lies in a lack of a labeled dataset as the training dataset for we have no idea how many unregistered NAT devices are in datasets or which of the hosts are NAT devices. As for host identification, regular IP address identity is impracticable due to the same source IP address modified by a NAT device. Elie Bursztein proposes a method to count the number of hosts behind one NAT device based on the TCP timestamp option [4]. Based on Elie Bursztein's research, Georg Wicherski has developed a method of host identification using TCP timestamp values [5].

Based on previous research, I perform the research in two stages, NAT detection and host identification. I first configure a local area network (LAN) connected to the Internet with a NAT device in our lab. End hosts on the internal network send out packets and receive packets through the NAT device, and I collect data on both interfaces of the NAT device. I collect a large amount of data and filter HTTP packets for NAT detection. Packets are then preprocessed by extracting features to build the behavior model. To create labeled data for supervised machine learning, artificial NATs are built according to Toma´s Kom˘arek's work. Three machine learning algorithms are used in this stage: Support vector machine (SVM), C4.5 and Alternating decision tree (AD Tree). I use an improved the feature merge method when generating artificial NATs instead of merely adding them up as in Toma´s Kom˘arek's work. In addition, I maintain the same proportion of different sizes of artificial NATs (the number of hosts of artificial NATs ranges from 5 to 15). NAT detection accuracy varies across different machine learning algorithms. Overall, the accuracy of NAT detection turns out to be quite high when I run the validation dataset.

Once the NAT device have been detected, the second stage is host identification. In this stage, I apply two different datasets and compare the results, one is the dataset I used in step one (HTTP dataset), the other is the TCP dataset filtered from original datasets I collected. Host identification is based on TCP timestamps, according to RFC 1323. TCP timestamp values must be at least approximately proportional to real time [6]. That is, the sets of TCP timestamp values and real time values (uptime of host) of packets from a host should display a linear line on a coordinate axis system. As different lines represent different hosts, we are able to know if packets are sent from the same host or not even their source IP addresses are the same.

Different from Georg Wicherski's real time work, our research is offline which then enables my proposed system to be used as a forensic analysis tool to analyze cybersecurity or other

incidents that could occur on an organization's network. I group packets by connection and build the lines by least-squares linear regression instead of calculating packet by packet. I managed to improve several aspects of the existing host identification methodology, including approaching host uptime instead of using system time, improving connection division, determining the boundary of number of packets in a connection to build the linear lines, correcting the computation method of distance of lines, and testing the threshold which is used to determine if two lines are close enough to belong to the same host. Apart from improving on methodology, I evaluate the experimental results in two ways to provide an overall evaluation (see Chapter 4). The experimental result of host identification is improved compared to that in Georg Wicherski's work. These two stages allow us to detect a NAT device from the Internet and then identify the hosts behind it.

In summary, the new contributions of this thesis research on NAT detection and host identification are listed as the following:

i. Evaluates different machine learning algorithms and demonstrates that AD Tree performs best in classification of NAT detection among the three classifiers;
ii. Identifies five effective attributes for NAT detection through experimental results;
iii. Improves on the artificial NAT generation method in building artificial NAT attribute vectors and handling inactive artificial NATs;
iv. Discovers that to achieve a high accuracy on NAT detection, a training dataset with the similar size to the target dataset is required;
v. Discovers the limit of the number of packets in one connection for host identification;
vi. Improves on the artificial line generation and distance calculation methods for host identification;
vii. Proposes and benchmarks a different evaluation method for host identification;
viii. Demonstrates that processing NAT detection and host identification separately can achieve a better performance than combining these two stages.

The rest of this thesis is structured as follows: chapter 2 summarizes related work on NAT counting, NAT detection, TCP timestamp option and host identification behind NAT devices. Chapter 3 discusses how I configure a NAT device and connect the internal network to Internet through it, it also shows how to generate traffic automatically, and discusses the methodology of how NAT detection and host identification work. Chapter 4 displays the experiments I

performed and analyzes the results. The last chapter draws the conclusions and discusses the future work.

Chapter 2 Related work

The previous chapter introduces the NAT, the threat it brings about, and difficulties in NAT detection and host identification. However, these are not the only problems with the NAT, for example, Nevena Vratonjic's research talks about threats of using NAT to users such as service providers learning about a user's location when the user is connected to public access points and generates location-based service (LBS) queries [7].

This chapter introduces previous work on NAT detection and host identification. Section 2.1 surveys different techniques of NAT device detection, including NAT behavior identification through traffic flows and NAT detection using HTTP access logs. Section 2.2 introduces approaches to count or identify hosts behind a NAT device. Finally, section 2.3 summarizes this chapter.

2.1 NAT detection literature review

Current NAT detection research is mainly based on NAT behavior modeling. Machine learning algorithms are generally adopted to build NAT behavior models from end hosts. Yasemin Gokcen' work focuses on exploring specific patterns in the network traffic that can identify NAT like behaviors [8]. His research uses machine learning approaches to automatically find patterns indicating NAT usage without using IP addresses, port numbers or payload information. The two machine learning methods in his research are C4.5 and Naive Bayes. He also uses a passive fingerprinting method that analyzes specific parameters without using features like IP addresses, port numbers and payload information in a given network traffic trace. An open source flow generator NetMate is used to generate flows. Features for passive fingerprinting like Time to Live (TTL) and HTTP User Agent String are then extracted from these flows. These features are employed by two classifiers: C4.5 and Naive Bayes. The research compares results of the two machine learning methods. It turns out that C4.5 learning classifier performs better than Naive Bayes, with a detection rate (DR) for both classes (class NAT and class others) of more than 95%, while the detection rate of class NAT using the Naive

Bayes classifier is less than 35%. So, in our research, C4.5 is also used in the NAT detection stage.

Toma´sˇ Koma´rek's research is also aimed at detecting NAT devices in the network via behavior. The behavior model in this research is built from IP-based features in HTTP access logs: number of unique contacted IP addresses, number of unique user-agents, number of unique OSs and versions, number of unique browsers and versions, number of persistent connections, number of upload bytes, number of download bytes, and number of sent HTTP requests [3]. In his work, artificial NATs are generated to label the training dataset for the supervised machine learning approach in order to deal with the problem of an insufficient number of labeled datasets. All the hosts are first labeled as end hosts and then artificial NATs are generated. Artificial NAT generation method is based on the fact that NAT gateways join traffic of multiple hosts into one without altering the eight features listed above. Hence the feature vector of one artificial NAT is the combination of feature vectors of all the end hosts it uses to generate the artificial NAT. In his work, all the feature vectors are simply added up to generate the artificial NAT feature vector. This method is improved on in our research in Section 3.2.2. As for machine learning algorithms, the support vector machine (SVM) is employed in Koma´rek's research due to its resistance to contaminated training datasets. As a result, those NATs mislabeled as end hosts have less impact on classification.

In Vojtech Krmicek's research, he proposed new approaches to NAT detection in three new fields based on Netflow: Time to Live (TTL), IP ID and TCP SYN packet size [9]. NetFlow is a Cisco network protocol to collect IP traffic information as well as monitor network traffic [10]. He managed to design a prototype NAT detection system, using multiple NAT detection techniques. Sebastian Abt's work is similar to that of Krmicek. It is also based on Netflow, and it is a passive remote NAT detection method based on behavior statistics from Netflow records [11]. In addition, there are some existing tools to detect NAT devices, such as NAT Classification/Detection Tool[1] using PJNATH (PJSIP NAT Helper Library), and Nat Probe[2] developed in Python.

[1] http://www.pjsip.org/pjnath/docs/html/group__PJNATH__NAT__DETECT.htm

[2] https://www.darknet.org.uk/2009/10/nat-probe-nat-detection-tool/

Our NAT detection stage is mainly based on the artificial NAT generation theory from Koma´rek's research. We improve on his artificial NAT feature vector generation method and perform research with different machine learning algorithms, including the SVM used in Koma´rek's research and the C4.5 used in Yasemin Gokcen' work, as well as Alternating decision tree (AD Tree).

2.2 Host identification literature review

In Napoleon Paxton's research, identifying network packets across translational boundaries, a packets identification approach based on payloads is proposed [12]. This approach relies on the fact that translational boundaries like NAT devices work by altering packet headers instead of packet payloads. In this research, the term "payload" is used as a unique identifier. This research applies cryptographic hashing techniques (MD5) to payloads of packets from both sides of the boundary, and then matches a packet before and after it is modified by a translational boundary. The first-in-first-out approach is used to match the encrypted payloads. Paxton's research allows us to match packets across NAT devices. However, it works on both sides of a NAT device and it is unable to identify end hosts behind a NAT device from the interface connected to the Internet.

In contrast, Sophon Mongkolluksamee proposes an approach to count end hosts behind a NAT device, which implements a per-flow IPID sequence; a random IPID; or a global IPID based on the sequence of IPID, a TCP sequence number, and a TCP source port in different manners [13]. IPID is a 16-bit counter that identifies unique packets when fragmentation occurs. Table 2.1 shows the different ways that different operating systems implement the three elements. Global IPID describes when all connections within the same host following a single sequence; per-flow IPID describes using a separate counter for each outgoing flow of packets; and random IPID uses a random number. The other two elements are described in table 2.1. The research processes a packet trace file to collect IPIDs, TCP sequence numbers, and TCP source ports of all packets, and then constructs sequences of them. The relationship among IPIDs, TCP sequence numbers and TCP source ports are classified to distinguish single hosts. Unlike Paxton's research, this research manages to count end hosts behind a NAT device, but it is unable to detect OpenBSD hosts which implement all the three elements randomly. In addition,

Mongkolluksamee's research can only count the number of hosts but cannot identify them from packets.

Table 2.1 Different ways of implementation on three elements by different operating systems

OS	IPID	TCP sequence number	TCP source port
Windows XP, Visa and 7	Global counter	Random number as starting sequence number for each TCP connection	Increase linearly proportionally to connection starting time
Linux 2.6	Per-flow counter	Counter	Counter
FreeBSD 8.1	Global counter	Random	Random
MAC OS 10.6	Random counter	Random	Increase linearly
OpenBSD 4.8	Random counter	Random	Random

In addition, Elie Bursztein's research, Time has something to tell us about network address translation, provides another approach to count end hosts behind a NAT device. Bursztein's research does similar things as Mongkolluksamee's; however, the method is totally different. Bursztein's approach is based on TCP timestamp options. According to RFC 1323, TCP timestamp value (TSval) must be at least approximately proportional to real time, in order to measure actual Round-Trip Time (RTT). The TSval can be described in equation 2.1, where $timestamp$ is the timestamp value, sec is the system uptime, $numincbysec$ is the increment rate, and is related to OSs, and in is an initial value.

$$timestamp = numincbysec * sec + in \qquad \text{Eq (2.1)}$$

The host counting approach relies on the fact that two hosts can't have the exact same system uptime unless they have been booted within the same millisecond and have the same OS. And the timestamp can rarely be the same within one short period. The linear functions of known hosts are stored in the form of two points of each function to verify each new TCP timestamp packet. If the new packet's TSval and uptime don't match all the known functions, it belongs to a new host. Because this is a theoretical research, it does not mention how a new function is calculated. Also, it can only count the number of end hosts behind a NAT device but cannot identify hosts.

Based on Bursztein's counting NATed hosts mechanism, Georg Wicherski introduces a technique of IP agnostic real-time traffic filtering and host identification using TCP timestamps

in his research. This research uses a variant function of equation 2.1, which is described in equation 2.2.

$t = a * s + b\ (ms)$ Eq (2.2)

In this equation, t is the TCP timestamp value, s is the Unix system time, a is the increment rate, and b is some initial value. Operating systems, with the exception of Windows, usually reset the value of TSval to zero at boot time, thus effectively setting b to $-a * s_0$, s_0 is the time of booting the system.

Artificial lines are generated to identify hosts. For every incoming TCP packet observed, the pair $(s_□, t_i)$of current system time $s_□$ and TCP timestamp $t_□$ is added to a list for that particular TCP connection. And when the connection terminates, a least-squares linear regression function is computed for the points $(s_□, t_i)$ collected for this particular connection. The computed (a, s_0) represents for a new host and is stored in the database. Whenever a new TCP packet is received, first, it is matched against existing hosts. Hosts are mapped into classes (different a's) to increase matching efficiency. The new packet is computed with each existing as for s_0, if it is close enough to a known function, it is considered to belong to the host. If it doesn't belong to any existing host, a new host is detected and the new (a, s_0) is stored in the database. Wicherski's work proposes a possible approach to identifying hosts behind a NAT device using TCP timestamp. The host identification stage of my research is based on his idea.

Other methods for host identification behind a NAT device are described as follows: Aniello Castiglione's research, Device tracking in private networks via NAPT Log analysis, determines a host profile or fingerprint to track down the device's movement and learn about the specific host behind one private network without its IP address [14]. In addition, Nino Vincenzo Verde proposes an approach to build a fingerprinting framework to identify users behind a NAT device using NetFlow records alone [15]. Hidden Markov Models (HMM) classifier is used in Verde's research to fingerprint users inside a network connected to the Internet through a NAT device. Different from the research introduced above, Rhiannon Weaver proposed an approach to visualize and model the scanning behavior of the conficker botnet in the presence of user and network activity in his work [16]. Weaver's work manages to count devices in an IP address space.

Therefore, host identification stage of my research is mainly based on Bursztein and Wicherski's research. It improves on Wicherski's methodology of host identification, including changes on the calculation method of host uptime values, the connection division method, the boundary of number of packets in a connection, the computation method of the distance of lines, the threshold which is used to determine if two lines are close enough to belong to the same host, and the evaluation methods of experimental results. In addition, the combination of the NAT detection stage and the host identification stage on the datasets in my research allows us to identify end hosts behind the NAT device detected on the Internet without knowing their private IP addresses.

2.3 Summary

In this chapter, both literature reviews of NAT detection and host identification are introduced. Koma´rek' NAT detection work, Bursztein's counting hosts behind a NAT device research and Wicherski's host identification behind a NAT device research all play important roles in my research. The next chapter will describe the methodology of the research in detail and how the three works are extended in my research.

Chapter 3 Methodology

This research is focused on TCP timestamp option based host identification behind a NAT gateway detected by supervised machine learning method from the network. In this chapter, first data generation details including network configuration and data collection are presented in Section 3.1. Then Section 3.2 introduces how I create artificial NATs to label the training dataset for supervised learning approach to detect NAT device from end hosts. The following Section 3.3 describes how I identify end hosts with the same source IP address behind a NAT gateway based on TCP timestamp option, as well as how I combine the two stages of the research in Section 3.4. The last Section 3.5 summarizes this chapter.

3.1 Data sets

3.1.1 LAN architecture

Data sets used in this research are collected from a local area network (LAN) connected to the Internet through a network address translation (NAT) device. The NAT device also works as a Dynamic Host Configuration Protocol (DHCP) server, which assigns IP addresses to 16 hosts on the LAN. Different operating systems and different browsers are installed to learn how the research works in different environments. The experimental configuration is shown in Figure 3.1.

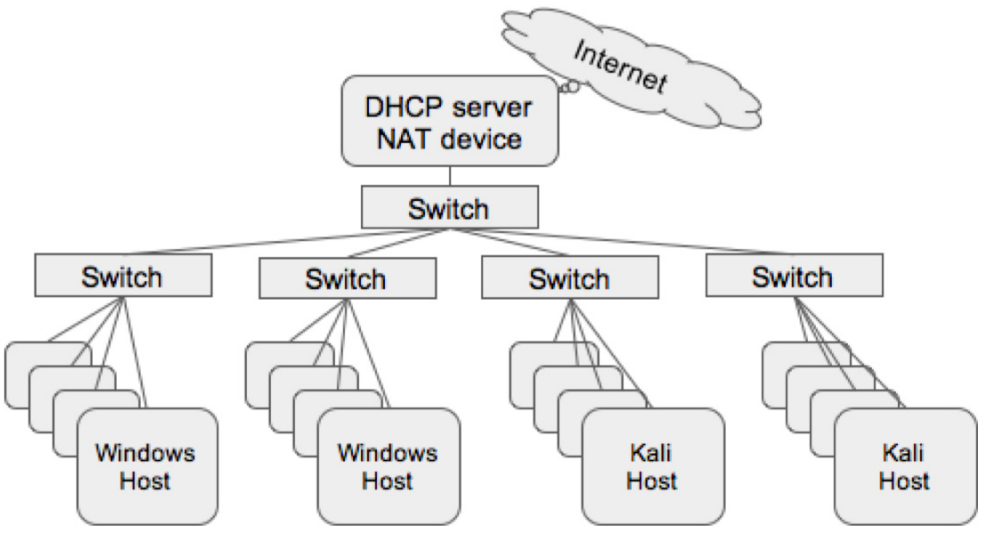

Figure 3.1 LAN with NAT device configuration

The host works as both a DHCP server and a NAT device; which is installed with Kali Rolling system. To configure the DHCP server and the IPv4 NAT gateway on the Kali host, we can follow the guide below[3]. The server host has two interfaces, one connected to Internet and the other connected to the hosts on the LAN. The DHCP server automatically assigns IP addresses to hosts on the LAN and allows them to connect to the Internet through the NAT gateway. The 16 hosts are grouped into four groups, and there are four hosts in each group connected to a group switch. In each group, hosts are installed with the same operating systems. Four group switches are connected to one switch which is connected to the server. The hosts system, IP address, interface and browser information are shown in table 3.1. The server has two interfaces, Eth0 connected to LAN, and Eth1 connected to Internet. Other hosts with two interfaces only have one interface working at the same time.

Table 3.1 Hosts and server information on LAN

Scen.	OS	Browser	Eth 0	Eth 1	Eth 2	Host name
0	Kali Rolling (NAT device + DHCP server)	Firefox	172.22.22.1 LAN	10.11.12.75 Internet	-	Yeti
1	Kali Rolling	Firefox	-	172.22.22.48	-	crocodile
2	Kali Rolling	Firefox	-	172.22.22.54	172.22.22.53	squirrel
3	Kali Rolling	Firefox	-	172.22.22.58	172.22.22.57	leopard
4	Kali Rolling	Firefox	-	172.22.22.56	172.22.22.55	pipingplover
5	Kali Rolling	Firefox	-	-	172.22.22.44	chipmunk
6	Kali Rolling	Firefox	-	-	172.22.22.45	alligator
7	Kali Rolling	Firefox	-	172.22.22.43	-	elephant
8	Kali Rolling	Google-chrome	-	172.22.22.51	172.22.22.42	panda

[3] https://codeghar.wordpress.com/2012/05/02/ubuntu-12-04-ipv4-nat-gateway-and-dhcp-server/

9	Windows 7 Professional	Google-chrome	172.22.22.28	172.22.22.32	-	orangutan
10	Windows 7 Professional	Firefox	172.22.22.33	172.22.22.26	-	gazelle
11	Windows 7 Professional	Firefox	172.22.22.29	172.22.22.25	-	giraffe
12	Windows 7 Professional	Google-chrome	172.22.22.31	172.22.22.30	-	polar bear
13	Windows 7 Professional	Firefox	172.22.22.99	172.22.22.40	-	ferret
14	Windows 7 Professional	Google-chrome	-	172.22.22.52	-	otter
15	Windows 7 Professional	Firefox	172.22.22.34	172.22.22.37	-	beaver
16	Windows 7 Professional	Google-chrome	172.22.22.35	172.22.22.38	-	groundhog

3.1.2 Data collection

Imacros [17], an extension for web browsers (both Google-Chrome and Firefox), is installed so hosts can automatically generate URLs and randomly download files. Imacros can be combined and controlled by JavaScript. I wrote a small program and ran it on Imacros. Figure 3.2 shows the flow chart of the Imacros program.

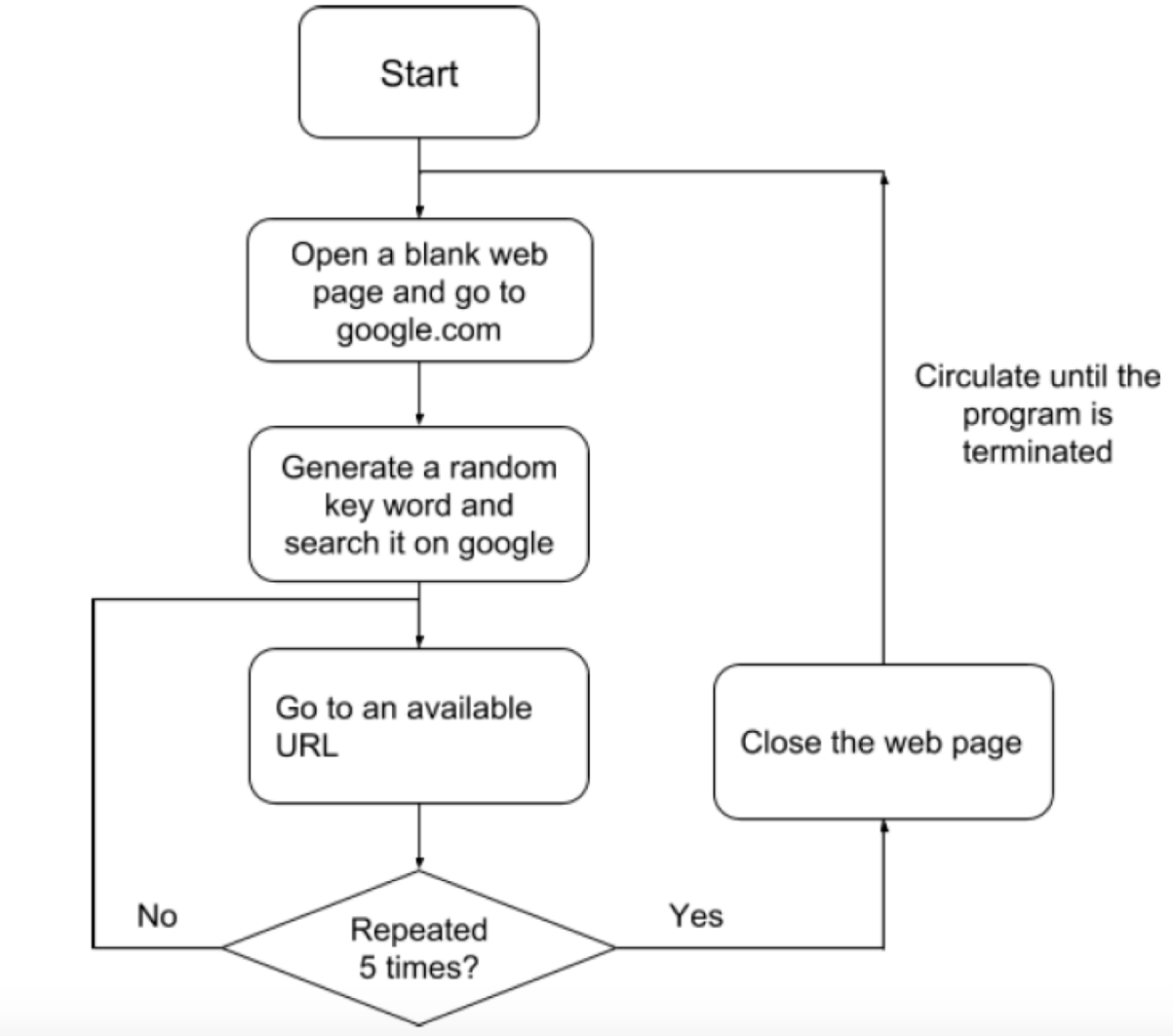

Figure 3.2 Imacros program flow chart

This program automatically searches random keywords on Google; it then randomly goes to one of the URLs, searches the URL for other URLs, randomly chooses one, goes to that URL and repeats the process several times. After this step, it closes the webpage and opens a new one to start the process again by searching for a random keyword on Google. The program circulates the above steps, and when it encounters files that can be downloaded, it downloads them. It will not stop until I terminate the program, or it is interrupted by system errors. Part of the program is shown below:

```
1. SET !ERRORIGNORE YES
2. SET !LOOP 2
3. SET !DATASOURCE_LINE {{!LOOP}}
4. TAB T=1
```

```
5. URL GOTO=https://www.google.com/?gws_rd=ssl
6. SET !VAR3 EVAL("var letters =
   ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','w','x','y','z']; var string = '';
   for(var i = 0; i < 4; i++){string += letters[parseInt(Math.random() * 25)]}; string")
7. TAG POS=1 TYPE=INPUT:TEXT FORM=NAME:f ATTR=NAME:q
8. CONTENT={{!var3}}
9. TAG POS=1 TYPE=BUTTON:SUBMIT FORM=NAME:f ATTR=NAME:btnG
10.     WAIT SECONDS=3
11.     TAG POS=1 TYPE=A ATTR=TXT:*w*b*
12.     WAIT SECONDS=3
13.     ONDOWNLOAD FOLDER=* FILE=* WAIT=YES
14.     TAG POS=2 TYPE=A ATTR=TXT:*Download*
15.     TAG POS=1 TYPE=A ATTR=TXT:*q*
16.     WAIT SECONDS=3
17.     ONDOWNLOAD FOLDER=* FILE=* WAIT=YES
18.     TAG POS=2 TYPE=A ATTR=TXT:*Download*
19.     TAG POS=1 TYPE=A ATTR=TXT:*y*
20.     WAIT SECONDS=3
21.     ONDOWNLOAD FOLDER=* FILE=* WAIT=YES
22.     TAG POS=2 TYPE=A ATTR=TXT:*Download*
23.     TAG POS=1 TYPE=A ATTR=TXT:*a*
24.     WAIT SECONDS=3
25.     ONDOWNLOAD FOLDER=* FILE=* WAIT=YES
26.     TAG POS=2 TYPE=A ATTR=TXT:*Download*
27.     WAIT SECONDS=3
28.     TAB CLOSE
```

This program starts by opening a web page and going to google.com. In line 6, the program generates a random four-letter word, and it searches the word on Google in line 7 and line 8. Line 10 through line 26 go to URLs five times, it is determined to go to URLs to make sure it doesn't end at a browser setting page or other pages that have no other URLs to go to. Every time it goes to a new URL, it downloads any available files. This program works a little differently on different operating systems: it creates more URLs on Kali hosts than on Windows hosts.

Hosts are grouped according to their operating systems in order to learn the differences of how Imacros runs on different systems and the impact on NAT detection and host identification. Also, different data from different systems can provide test datasets that are different from training datasets. Different combinations of groups run in 17 days. Data collected are stored by

different combinations. Hosts running schedule is shown in Table 3.2. Hosts with marks are running and generating data on the marked date, and hosts without marks are powered off on that date.

Table 3.2 Hosts running status by date

OS	Browser	Running status by date					
Kali Rolling (NAT device)	Firefox	Aug01	Aug02-Aug13	Aug14	Aug15	Aug16	Aug17
Kali Rolling	Firefox	√	√	√		√	
Kali Rolling	Firefox	√	√	√		√	
Kali Rolling	Firefox	√	√	√		√	
Kali Rolling	Firefox	√	√	√		√	
Kali Rolling	Firefox	√	√	√			√
Kali Rolling	Firefox	√	√	√			√
Kali Rolling	Firefox	√	√	√			√
Kali Rolling	Google-chrome	√	√	√			√
Windows 7 Professional	Google-chrome	√	√		√		√
Windows 7 Professional	Firefox	√	√		√		√
Windows 7 Professional	Firefox	√	√		√		√
Windows 7 Professional	Google-chrome	√	√		√		√
Windows 7 Professional	Firefox		√		√	√	
Windows 7 Professional	Google-chrome		√		√	√	
Windows 7 Professional	Firefox		√		√	√	
Windows 7 Professional	Google-chrome		√		√	√	

Data is collected on NAT device's both interfaces, LAN and Internet interfaces, for two phases of research. In total, I collected 521 GB of data. The data size by date is shown in Table 3.3. Tcpdump is employed to capture packets and build pcap files.

Table 3.3 Data size by date

	Data size (GB)		
Date	eth0 - LAN interface	eth1 - Internet interface	total
Aug 01	12	11	23
Aug 02	23	21	44
Aug 04	56	54	110
Aug 08	28	27	55
Aug 09	19	18	37
Aug 10	30	28	58
Aug 11	46	42	88
Aug 14	22	20	42
Aug 15	7	6	13
Aug 16	10	9	19
Aug 17	17	15	32
Total	270	251	521

3.1.3 Data sets

Tshark [18] is exploited to extract information packet by packet from raw data in both phases of the research. Csv files are generated for further processing. Tshark is a network protocol analyzer; it is a terminal oriented version of Wireshark designed for capturing and displaying packets. It can deal with large size pcap files by running terminal commands without displaying all the packets as Wireshark does. I can select relevant packet features and save them into csv files.

In the first stage, data is from NAT device's Internet interface. Packets are captured from LAN through the NAT device, whose source IP addresses are translated into the same one (NAT device's IP address). Packets from Internet are also captured. This stage is aimed to distinguish NAT devices from end hosts on the Internet. Packet information is extracted from HTTP headers. Necessary packet features and corresponding tshark options are shown in Table 3.4.

Table 3.4: HTTP features and tshark options

Packet features	Tshark option

Packet number	frame.number
Host's IP address	ip.src
Server's IP address	ip.dst
HTTP status	http.connection
URL of request	http.request.uri
User-agent information	http.user_agent
Sum of download/upload bytes	http.content_length
Ending time of communication and its duration	http.time
HTTP method	http.request.method

The second stage is aimed to distinguish different end hosts through packets translated by a NAT device, although their source IP addresses are the same. In this stage, data from the NAT device's Internet interface is used as a training set and data from the LAN interface is used as a validation set. Packets from the LAN interface are directly sent from end hosts inside the LAN (before network address translation), whose source IP addresses are original. Matching packets from both interfaces provides validation sets for this phase's research. Packets information is extracted from TCP headers. Necessary packet features and corresponding tshark options are shown in Table 3.5.

Table 3.5 TCP features and tshark options

Packet features	Tshark option
Packet number	frame.number
System time	frame.time
Source IP address	ip.src
Destination IP address	ip.dst
Source port number	tcp.srcport
Destination port number	tcp.dstport
TCP Timestamp value	tcp.options.timestamp.tsval
TCP FIN flag	tcp.flags.fin
TCP reset flag	tcp.flags.reset

3.2 NAT detection

The NAT detection mechanism is based on NAT device behavior analysis; this uses supervised machine learning for binary classification. It aims to differentiate NAT devices from end hosts in the network, followed by next stage host identification. These two steps allow us to identify different hosts even behind a detected NAT device in the network.

3.2.1 Host behavior vector

The behavioral features differentiate a NAT device from end hosts. A NAT device contains multiple end hosts and modifies host IP address as well as server IP address in IP headers and then transits modified packets to or from end hosts. Hence the communication volume of a NAT device is several times that of an end host depending on the number of end hosts behind a NAT device. Packets from all hosts are captured in one day as mentioned in the dataset section to fairly measure the volume. To learn NAT device behavior, I first extract host information from HTTP packets. Packet features I extract include IP source address, IP destination address, HTTP connection status, URL of HTTP request, user-agent information, sum of download bytes and upload bytes, communication duration and HTTP method. Those features are obtained with tshark from packets headers; corresponding tshark options are listed above.

Packets are represented by features, and next I count those features for each host. Here I take unique IP addresses as hosts, which can be a NAT device or an end host. Features extracted from each packet are counted as follows [3]:

1) number of unique contacted IP addresses
2) number of unique user-agents
3) number of unique OSs and versions
4) number of unique browsers and versions
5) number of persistent connections
6) number of upload bytes
7) number of download bytes
8) number of sent HTTP requests

Operating system information and browser information are included in user-agent strings from HTTP request headers. While HTTP request methods indicate the packet download or upload data. Persistent connections are connections with keep-alive headers. These features reflect the communication volume of hosts and vary from NAT devices to end hosts. Another element that affects the features is how long the host is active, or how often is the host active, including both end hosts and NAT devices. Theoretically, a NAT device transports much more packets than an end host does, but if the NAT device is not active, its traffic volume can be less than an end host's. This requires that the machine learning algorithm should be resistant to contaminated training datasets. I record each host with a vector of the eight features above and then apply it to machine learning to build behavior models for NAT devices and end hosts, so that it can be applied to classify NAT devices and end hosts later.

3.2.2 Artificial NAT

To prepare training datasets for machine learning, a labelled dataset is required. But I have no idea how many and which of the hosts detected on the Internet are NAT devices or end hosts. Considering the fact that the proportion of NAT devices in all the hosts is negligibly small, I employ the artificial NAT generation method in Komaŕek's research [3]. That is to label all the hosts as end hosts and create artificial NAT devices from those end hosts. Labelled artificial NATs are then added to the training dataset so that it has both class NAT and class end host. There is no doubt that this will causes deviation to experiment results, albeit to a small degree. This also requires the machine learning algorithm I use being immune to contaminated datasets. Machine learning algorithms will be introduced in the next section.

Artificial NAT device generation is based on the fact that a NAT device will not change the eight features mentioned above when modifying IP addresses. Hence a NAT device's features are the combinations of features of end hosts behind it respectively. In Tomaś Komaŕek's work, a feature vector of a NAT device is calculated as the sum of its end hosts' feature vectors [3]. It works for most of the features, but for features like number of unique user-agents, number of unique OSs and versions and number of unique browsers and versions, it makes no sense. Those three features cannot be simply added up. For example, a NAT device has three end hosts with 1, 1, 2 unique operating systems respectively, and each

end host is installed with the same version of Windows system, the last one is installed with the same Windows system as well as a Linux system. This NAT device's number of unique OSs should be two instead of four, which is the sum of end hosts' unique OSs. Similarly, the other two features of a NAT device are not the sum of its end hosts' features. In our work, an artificial NAT device's number of unique user-agents, number of unique OSs and versions and number of unique browsers and versions are calculated as the maximum numbers of its end hosts' corresponding features. It is more reasonable than simply adding up all the hosts' features.

The number of end hosts behind an artificial NAT ranges between 5 and 15. To balance different sizes of NAT devices, I create artificial NATs with a number of hosts from 5 to 15 as a unit. The number of NAT units depends on the number of detected hosts. Each end host is used once in composition of artificial NATs, and the number of NAT units is the maximum number all the hosts can generate. Our artificial NAT device generation method generates the same number of NATs with 5 to 15 end hosts. And the total amount of NATs is decided by the number of hosts; the more hosts detected, the more artificial NATs are generated. Training dataset then can be more reasonable than building a united number of artificial NAT devices under all circumstances.

3.2.3 Machine Learning and Classification

As mentioned above, it is vital for the NAT device detection algorithm to be resistant to contaminated datasets. In addition, host classification is based on linear combination of features. Based on the above limitations, I apply the training dataset to three kinds of machine learning (ML) classifiers to compare the results. The following three ML algorithms are applied in Waikato Environment for Knowledge Analysis (Weka), a suite of multiple machine learning software providing easy access to both classification and analysis.

a) Support vector machine

Support vector machine (SVM) is a supervised machine learning model, a non-probabilistic binary classifier [19]. It performs both linear classification and non-linear classification. It performs efficient classification and evaluation. Also, SVM tends to be resistant to

contaminated training datasets, and our manually labelled training dataset contains some wrongly labelled records. It is one of the most appropriate machine learning algorithms for our work.

SVM is about looking for the "maximum-margin hyperplane" to divide points (our samples) into two classes, when the distance of hyperplane and nearest point of each class is maximized. Here is how linear SVM works. Data items in the training dataset to be processed are in the form of

$$(\vec{x_1}, y_1), \ldots\ldots, (\vec{x_n}, y_n)$$

where a point$i$, in the form of $\vec{x_i}$ is a p-dimensional vector, y_i is a value 1 or -1 and represents two classes of all data points. If our data items have p features, I plot a p-dimensional space using all the records, and the points are represented by vectors with feature values. Then I try to find out the best $(p-1)$-dimensional hyperplane to separate the points. A hyperplane can be written in the form of equation 3.1:

$\vec{w} * \vec{x} - b = 0$, Eq (3.1)

where $\vec{w}$ is the normal vector of the hyperplane. The distances of hyperplane and nearest point from each class are calculated, they are called Margin. If the dataset is linear separable, the distance of two parallel hyperplanes that separate each class can be largest, as is shown in Figure 3.3. Then the best hyperplane is the hyperplane right in the middle of them.

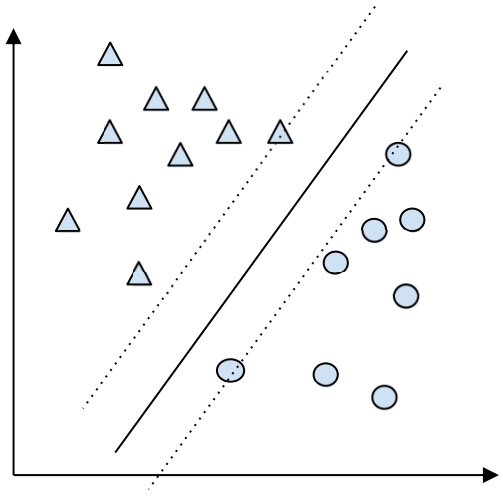

Figure 3.3 Linear SVM example

These two classes separated by the two hyperplanes can be described as equation 3.2 and equation 3.3:

$\vec{w} * \vec{x_i} - b \geq 1$, if $y_i = 1$, Eq (3.2)

any points on or above this hyperplane is classified as class 1;

$\vec{\omega} * \vec{x_i} - b \leq -1$, if $y_{\square} = -1$, Eq (3.3)

any points on or below this hyperplane is classified as class -1. The distance of two hyperplanes is: $\frac{2}{\|\vec{\square}\|}$, to maximum the distance is to minimize $\vec{\square}$. Then the two inequalities above can be written as equation 3.4.

$y_{\square}(\vec{\omega} * \vec{x_{\square}} - b) \geq 1$, for all $1 \ll i \leq n$. Eq (3.4)

Then to find out the best hyperplane with maximum Margin is an optimization problem as equation 3.5.

Minimize $\vec{\square}$ subject to $y_{\square}(\vec{\omega} * \vec{x_{\square}} - b) \geq 1$, for $i = 1, \ldots, n$. Eq (3.5)

SVM selects the hyperplane with maximum Margin on condition that it classifies the classes accurately.

As is shown in Figure 3.4, this algorithm has a feature of ignoring outliers. Triangles and circles are two different classes here. The hyperplane in the figure classifies most points into correct classes and the distance of it to nearest points from each class is maximized. Although there is one triangle point classified wrongly into another class, it still selects the best hyperplane to classify other data points. This feature is important to our experiment for there are unknown NATs labelled as end hosts in our training datasets.

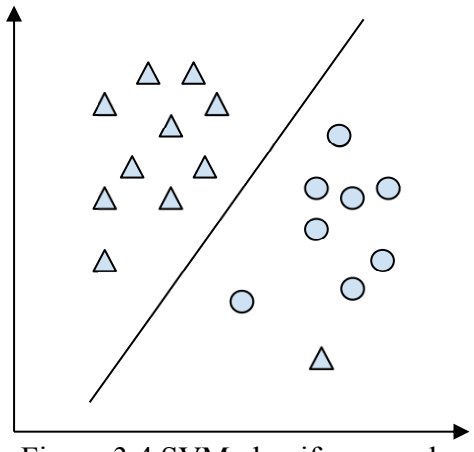

Figure 3.4 SVM classify example

b) C4.5

Another algorithm I used in our experiment is C4.5, a commonly used machine learning algorithm that builds decision trees from classified training dataset [20]. It chooses the attributes that most effectively split samples into different classes as the nodes to build the tree.

C4.5 is an improved version of ID3 algorithm; it handles missing attribute values in training dataset. It generates a more effective and accurate decision tree for our experiment.

ID3 is a well-known machine learning algorithm to generate a decision tree to classify data items [21]. It first calculates the information gain from entropy of each attribute from the target dataset. Entropy $H(S)$ is shown in equation 3.6 and is a measure of how much uncertainty the dataset has.

$H(S) = \sum_{x \in X} -p(x) log_2 p(x)$ Eq (3.6)

where S is the target dataset, X is set of classes in dataset S, x is a class in X , and $p(x)$ is the proportion of the number of data items in class x to the number of all items in dataset S. Information gain $IG(A,S)$is shown in equation 3.7 and is calculated from the entropy.

$IG(A,S) = H(S) - \sum_{t=T} p(t)H(t)$ Eq (3.7)

where T is the subsets split from dataset S by attribute A, $S = \cup_{t \in T} \quad t$, $p(t)$ is the proportion of number of data items in subset t to the number of items in target dataset S and $H(t)$ is the entropy of subset t. And then the dataset S is split into subsets according to the attribute which makes the resulting entropy minimum. This attribute is one node of our decision tree, the rest of the attributes are selected in subsets recursively.

C4.5 algorithm is different from ID3 in that it can handle continuous attribute values as well; it creates a threshold and splits the values by comparing them with the threshold. C4.5 is also robust to missing attribute values and ignores missing values in gain and entropy calculations. The most important improvement of C4.5 is that it prunes trees after creation, which effectively reduce the useless leaf nodes, leading to an efficient decision tree. C4.5 helps us to focus on the attributes that play a vital role in NAT detection.

c) Alternating decision tree

The last ML algorithm in our experiment is Alternating decision tree (AD Tree) [22]. It also generates decision trees as C4.5. But its nodes generation method is entirely different from the C4.5 algorithm. AD tree considers all the features when classifying instances, and it also works well in our experiment.

An AD tree consists of two types of nodes, decision nodes and prediction nodes. Two classes are coded into 1 and -1 respectively. Data items in the training dataset to be processes are in the form of

$$(x_1, y_1), \ldots\ldots, (x_n, y_n)$$

where $x_{\square}$ is a vector of attributes, $y_{\square}$ is the class of data items, and it is either 1 or -1. Decision nodes of AD Tree contain a prediction condition of attributes from the target dataset, and prediction nodes contain a positive or negative number, indicating how the condition impacts final classification. The number in prediction nodes is weight $\omega_{\square}$. A data item should go through every path of the decision tree and compare its attribute values with each decision condition to get value from prediction node. All the prediction values are then added up, it is the final score of this data item as shown in equation 3.8.

$W(x_{\square}) = \sum_{d \in D} \omega_i (d)$ Eq (3.8)

where d is prediction condition from decision node and $\omega_{\square}(d)$ is the wieght of data item $x_{\square}$ on prediction condition d. Whether the final score $W(x_{\square})$ is a positive number or negative one decides this instance's class. If the final score is a positive number, then its class is the class coded as 1 and vice versa.

AD tree is different from all the other decision trees in that each instance should go through every path of the decision tree, and all the values from the prediction nodes it passes are calculated as the final score. This algorithm takes every possible condition into consideration, and this gives us a precise classification of NAT detection.

These three machine learning algorithms I apply to our datasets study different aspects of the NAT detection. SVM is the algorithm used in Toma´sˇ Koma´rek's work, C4.5 reveals the vital attributes that influence the classification, and AD Tree provides a precise classification from every possible effective attribute. Different experiment results on these three algorithms are shown in the next chapter.

Training datasets for machine learning are generated by creating artificial NATs as mentioned above. I use a certain percentage of training datasets as test datasets, as well as new validation datasets as test datasets. The new validation datasets or test datasets I use are also captured from our lab, on both Internet interface and LAN interface. But those datasets exclude unknown hosts from the Internet, that is, it contains the real NAT device and other end hosts in our lab.

According to our artificial NAT generation method, this real NAT device in the training dataset is labelled as end host. If the real NAT device is then classified as NAT device in validation step, it supports the idea that instances that wrongly labelled as end hosts in training datasets do not affect experiment results.

Our algorithm improves Tomaˊsˇ Komaˊrek's work in generating artificial NATs, does research in NAT detection in different aspects with different machine learning algorithms, and provides a more convincing validation process. Steps of NAT detection can be summarized in Figure 3.5.

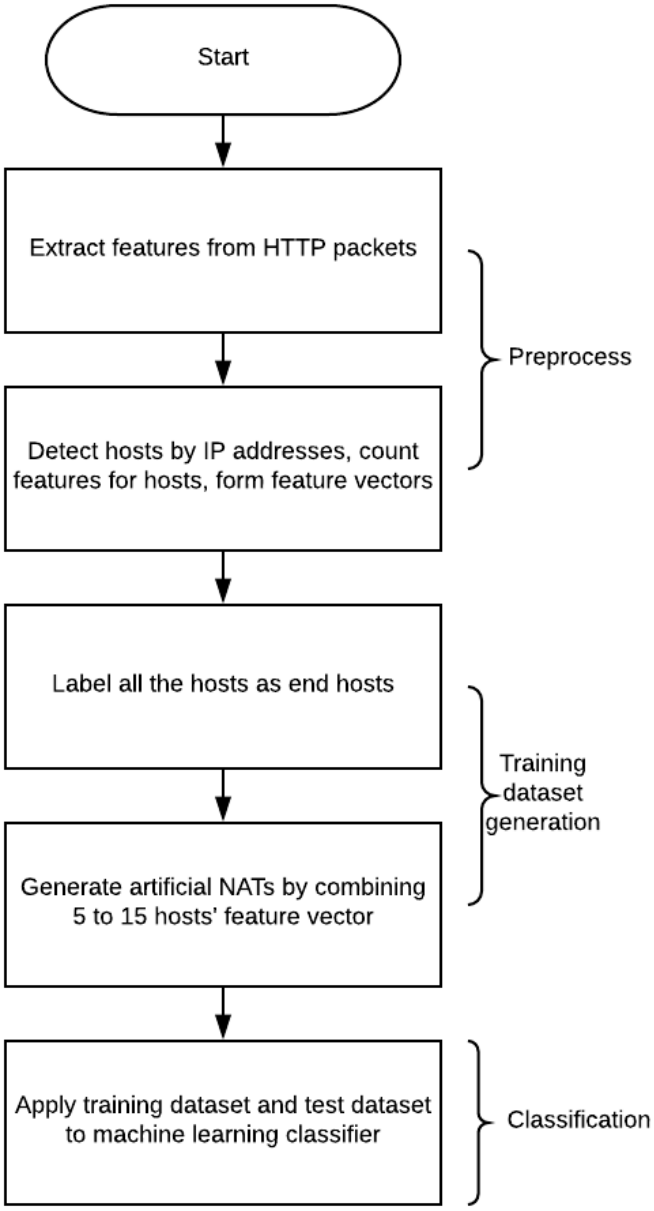

Figure 3.5 NAT detection flow chart

3.3 Host identification

3.3.1 TCP timestamp function

Host identification behind a NAT device is based on features of TCP timestamp. TCP timestamp option is inside TCP header's option field. In RFC 1323 [6], TCP timestamp value (TSval) is defined as follows.

The timestamp value to be sent in TSval is to be obtained from a(virtual) clock that I call the "timestamp clock". Its values must be at least approximately proportional to real time, in order to measure actual RTT.

Timestamp is an affine-linear function as shown in equation 3.9 [5].

$timestamp = numincbysec * sec + in\ (ms)$ Eq(3.9)

where $numincbysec$ is the increment rate of "timestamp clock", in is initial timestamp value, sec is the system uptime. The initial number in is zero for all operating systems except Windows, it is random for Windows. RFC also mentions that the increment rate $numincbysec$ should between 0.001 (1 tick per millisecond) and 1 (1 tick per second).

The assumption of the pair of ($numincbysec$, in) can uniquely identify an end host is based on the idea that any two hosts behind a NAT device can not have the same uptime unless they Ire booted at the same time in milliseconds.

In Georg Wicherski's work, this function is mentioned as equation 3.10 [5].

$t = a * s + b\ (ms)$ Eq (3.10)

where a is the tick scale of the timestamp clock and b is some initial value, s is a UNIX timestamp (the number of milliseconds elapsed since midnight of January 1, 1970(UTC)). For this function, the pair of (a ,b) uniquely identifies an end host until rebooted.

Comparing the two functions above, I found that equation 3.10 is a variant of equation 3.9. Tick scale a in equation 3.10 and increment rate $numincbysec$ in equation 3.9 are the same, while independent variable s in equation 3.10 is system time (UNIX timestamp), but sec in equation 3.9 is system uptime. From packets information collected, system time is available, but system uptime is unknowable. The relationship between these two variables is shown in equation 3.11.

$s = sec + boottime\ (ms)$ Eq (3.11)

where $boottime$ is the number of milliseconds elapsed since midnight of January 1,1970(UTC) until the host were booted. Equation 3.10 can be converted as equation 3.12.

$t = a * sec + a * boottime + b\ (ms)$ Eq (3.12)

Wicherski's function approximates equation 3.9, but they are not equivalent, which might be one of the reasons why there are errors in their experiment results. So, in our work, I subtract a uniform value from system time to make it near to actual system uptimesu. And I replace system time s in equation 3.10 with simulated system uptime su. The calculation of this uniform value is based on the fact that increment rate of timestamp value should between 0.001 and 1 mentioned in RFC1323.

3.3.2 Host identification algorithm

The basic idea of identifying hosts behind a NAT device using timestamp values is as follows. For each active connection, each pair of ($su_□$, $t_□$) of simulated system uptime su_i, and timestamp value t_i of packet i is added to a particular list. Then a least-squares linear regression function is computed for the list. This algorithm computes (a,b) for each TCP connection, which makes the sum of squares $\sum_{i=1}^{m} \epsilon_i^2$ of the error $\epsilon_□ = |t_i - (a * su_i + b)|$ is minimized. This pair of (a,b), or the line that is defined by function $t = a * su + b$, uniquely identifies a host. All the pairs are saved in our database, and everytime a new pair of (a,b) is computed, I first try to match it against existing pairs in the database. Matching pairs here means to find out if distance between the two lines is small enough. If a match is found, it can be concluded that this connection is from the detected host in our database. If this pair has no match, then it comes from a new host, and I'll add this pair to the database.

The algorithm matches connections, which provides more information than matching single packets. It can be more reliable and efficient. In Wicherski's work, a 4-tuple ($src_{□□□□}$, $src_{□□□□}$, $dest_{□□□□}$, $dest_{□□□□}$) is referred to as one connection until a packet with RST flag or FIN flag is detected. This method works for normal circumstances. But only packets with TCP timestamp values are supported for this experiment, and as mentioned above, TCP timestamp values are optional. The deficiency of packets without TCP timestamp values causes errors in connection recognition and then affects final results. In our work, I use tshark to separate

packets by connection and then deal with connections by connection sequence. Also, due to the deficiency of packets without TCP timestamp values, the number of packets in one connection is not enough to compute an accurate line for the connection. I set up experiments to test how different numbers of packets in connections impact the accuracy of identifying end hosts. The accuracy is enhanced with the increase of number of packets in one connection. I adopt 55 as the minimum number of packets in one connection in our experiment according to our test (see next section).

All the end hosts are saved in our database. Whenever a new connection is processed, the pair of (a,b) computed for it will be compared with all pairs of (a,b) in the database, which is not efficient. To improve the efficiency of matching the lines computed for connections, or pairs of (a,b), I first match a. As mentioned above, a is the increment rate of timestamp values, and it is related to OS implementation. As for the line, a is the slope, that is, same a's means parallel lines. If the difference between two a's is smaller than a certain threshold $\delta_{\square\square\square\square}$, they are identified as one slope, that is to say, the two lines are parallel. In Wicherski's work, he uses a default value of $2ms$ as the threshold without explaining why. I test different values of threshold to see its influence in our work. Different hosts are classified by slopes, and they are saved in lists, so that for each new line, it is not necessary to compare every host in the database. If the new connection's a is related to an existing slope, I compute the distances of all the hosts' lines of this slope with the new line. If distance is below threshold $\delta_{\square\square\square\square}$, this connection is associated to the existing host. If all the hosts of this slope do not meet the criteria, the new line is added to this slope's list, it belongs to a new host. If the connection's slope is not found, it also belongs to a new host. In that case, I create a list for the new slope, and add the line to the list.

The lines I store in the database are in the form (a,su_0), where su_0 is the solution of function $t = a * su + b$ when $t = 0$. As stated above, a, the increment rate of timestamp value, is between 0.001 and 1 ($a > 0$). When $t = 0$, $su_0 = \frac{-b}{a}$. The solutionsu_0 is the system uptime when timestamp value is zero, which can be predicted by least-squares linear regression algorithm. In Wicherski's work, when a slope's match is found in the database (the distance between $a_{\square}$ and one $a_{\square}$ in database is below threshold $\delta_{\square\square\square\square}$), he tried to find the closest s_0(corresponding to su_0 in our work) to match the line. Nevertheless, the distance between twos_0's or twosu_0's is different from the distance between two lines. The distance between

two parallel lines $a * su - t + b_1 = 0$ and $a * su - t + b_2 = 0$, is $d = \frac{|b_1 - b_2|}{\sqrt{a^2 + (-1)^2}}$. The distance between two parallel lines is $d = \frac{|b_1 - b_2|}{\sqrt{a^2 + 1}}$, while the distance between two s_0's or su_0's is $d_{\square} = \frac{|b_1 - b_2|}{a}$. When a is smaller than 1, d and $d_{\square}$ can differ by orders of magnitude. Under this circumstance, threshold $\delta_{\square\square\square\square}$ cannot measure distance. Since the distance between lines d and the distance between two s_0's or su_0's $d_{\square}$ are interralated, I can calculate d from $d_{\square}$: $d = \frac{d_{\square} * a}{\sqrt{a^2 + 1}}$. Then d is compared with threshold $\delta_{\square\square\square\square}$ to determine whether this connection is close enough to the line of one known host or not as stated above. The steps to identify hosts behind a NAT device are summarized in the flowchart shown in Figure 3.6.

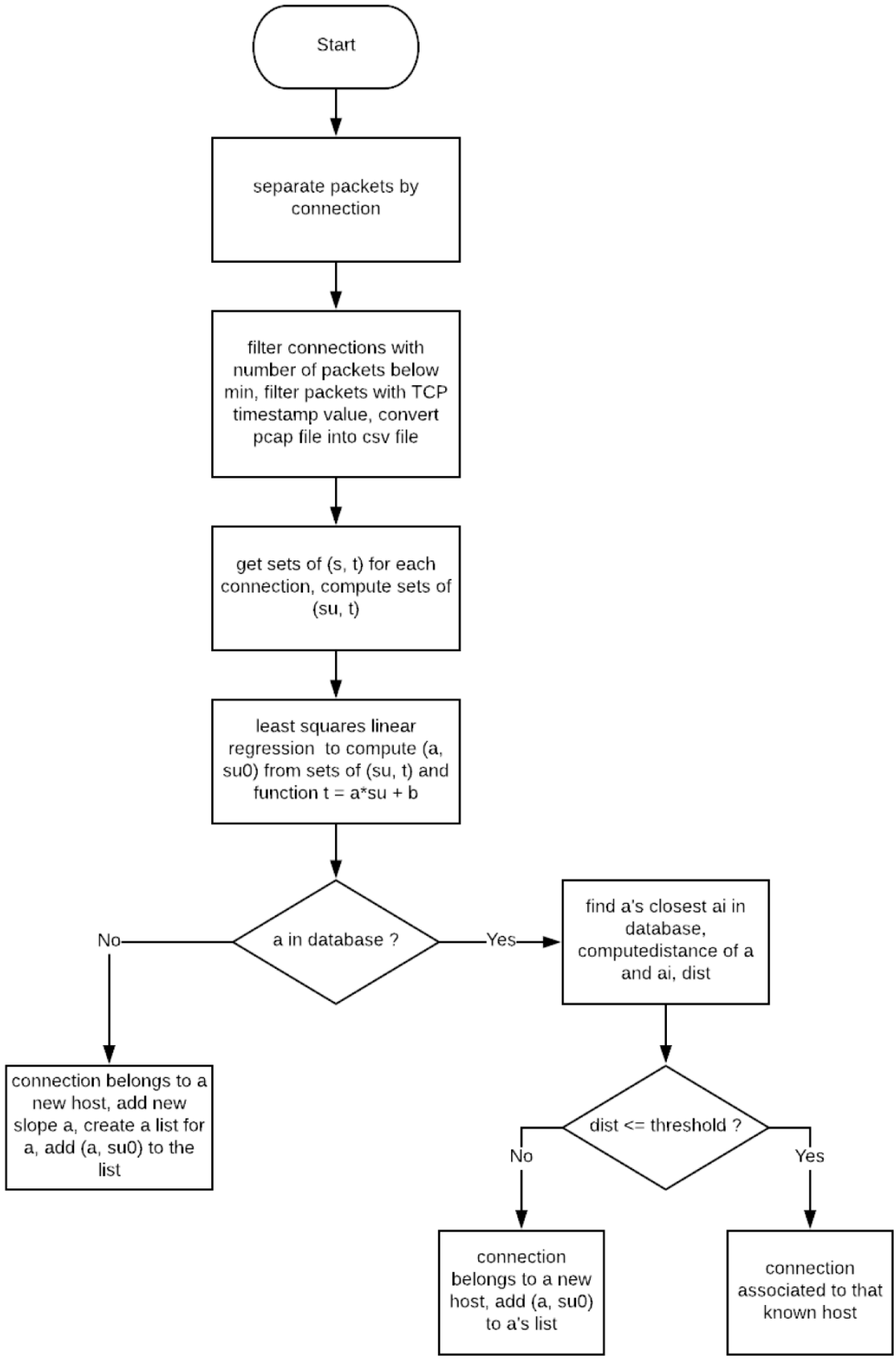

Figure 3.6 Host identification flow chart

This approach improves host identification through TCP timestamp in practice. I managed to approach host uptime from system time, improve connection division and find the boundary of number of packets in one connection when building the artificial lines to identify end hosts, learn the impact of different values of threshold to identify if two lines are close enough or if one connection is associated to a known host, and correct the computation method of the distance between lines. This leads to significant improvement of our experiment results in identifying end hosts with TCP timestamp values.

3.4 NAT detection and Host identification

The above two sections introduce two stages of our experiment separately: NAT detection and host identification. As is shown in Figure 3.7, I perform the two steps in order.

First, HTTP packets are filtered from the original dataset, then I extract HTTP attributes from the filtered dataset for NAT detection. Once the NAT device is detected through the machine learning method, packets from the NAT device are then filtered out for further processing. Only packets with a TCP timestamp option are selected for host identification as mentioned above. Apart from improvements on both of the steps mentioned in previous sections, processing host identification phase immediately following NAT detection phase allows us to identify end hosts behind a detected NAT device even though packets are from the same IP addresses.

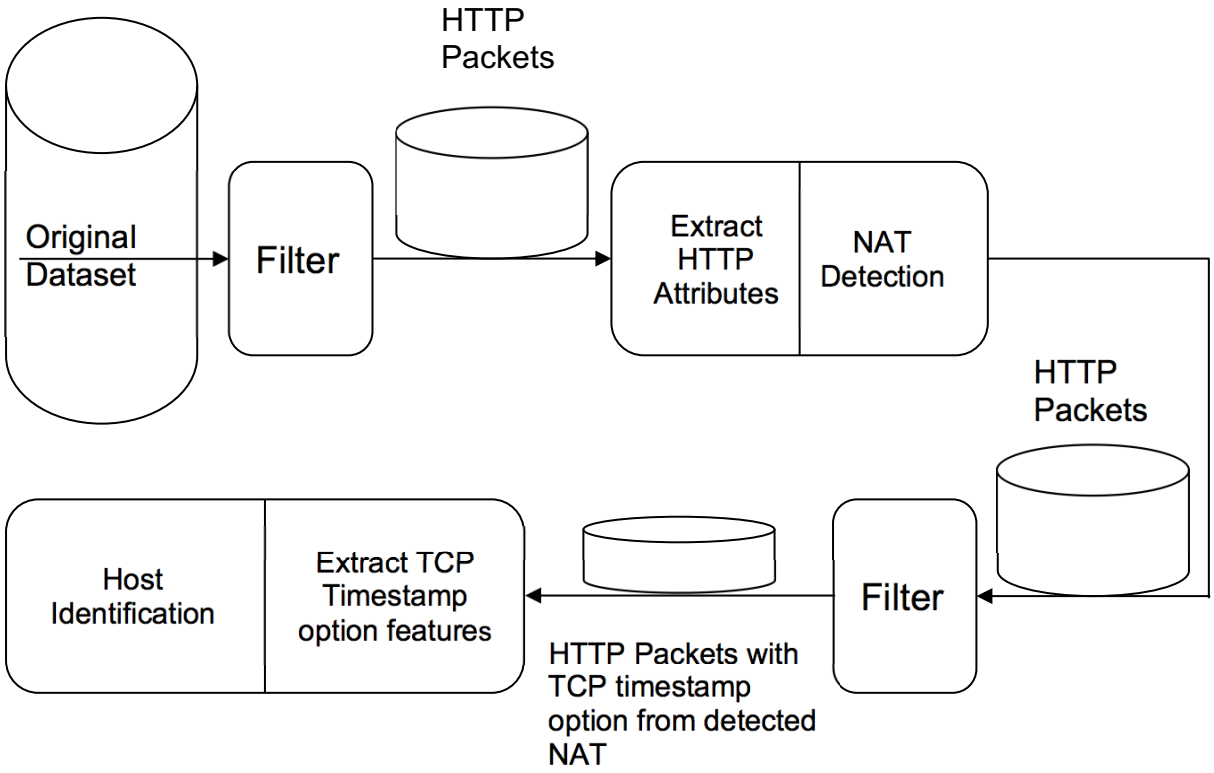

Figure 3.7 NAT detection and Host Identification

3.5 Summary

In summary, the research begins at data collection step. Collected datasets are preprocessed to extract HTTP features for NAT behavior model building in NAT detection supervised learning step, and then datasets are processed to extract TCP features for host identification by creating artificial lines of TCP timestamp value and system uptime. These stages in our research are used in experiments in the following chapter.

Chapter 4 Experiments and Evaluation

In this chapter, dataset selection from all the data collected is explained in Section 4.1. Supervised machine learning experiments and results of NAT detection are described in Section 4.2. Section 4.3 introduces the host identification experiments and evaluation on the experiment results using a Python program. Experiments of host identification using datasets of NAT detection stage and their results are displayed in Section 4.4. Section 4.5 summarizes this chapter.

4.1 Dataset selection

Datasets for both NAT detection stage and host identification stage were captured from the internal interface and the Internet interface of the LAN introduced in Section 3.1.1. All the datasets collected in 17 days are too large to be completely processed, at a size of 521 GB. Hence five days' datasets were selected as target datasets:

a) Aug 02: all the hosts on the LAN running
b) Aug 04: all the hosts on the LAN running
c) Aug 14: only Kali hosts on the LAN running
d) Aug 15: only Windows hosts on the LAN running
e) Aug17: half of the Kali hosts and half of the Windows hosts on the LAN running

Datasets from these five days were collected with different combinations of hosts running. Performing experiments on datasets with different operating systems examines the generalization of the methodology and provides comparison of the impacts of different systems on the NAT detection and the host identification.

4.2 NAT detection experiment

In this section, experiments and results of three machine learning classifications on both test datasets and validation datasets are described. Each test dataset is 20 percent of the training

dataset; this training dataset is labelled using the method introduced in Section 3.2.2. Since training datasets were labelled manually, I also apply the three classifiers on each validation dataset. The validation datasets include the real NAT device in the lab.

4.2.1 Training and test

After preprocess as mentioned in the last section, I applied machine learning algorithms in Weka to training datasets with labels. The first algorithm I used was Support Vector Machine (SVM). I used 80 percent of the preprocessed Aug 02 dataset as the training dataset and used the remaining 20 percent as the test dataset. Results of the experiment are shown in Table 4.1. In total, 1534 instances out of 1568 were classified correctly, a high accuracy of 97.83%. There are 1437 end host instances and 131 NAT device instances in the test dataset. Table 4.2 shows the confusion matrix of the SVM results. There were 31 actual NAT devices incorrectly classified as end hosts, and 3 actual end hosts classified as NAT devices. So, the true positive rate (TPR) of class end host was 1434/1437, 99.79%, and the TPR of class NAT device was 100/131, 76.34%, which was lower than that of class end host.

Table 4.1 SVM results using 20% as test dataset

		Proportion
Correctly Classified Instances	1534	97.83%
Incorrectly Classified Instances	34	2.17%
Kappa statistic	0.8432	
Total Cost	34	
Average Cost	0.0217	
Mean absolute error	0.0217	
Root mean squared error	0.1473	
Relative absolute error	13.62%	
Root relative squared error	53.20%	
Total Number of Instances	1568	

Table 4.2 SVM confusion matrix

Confusion Matrix		
a	b	<-- classifies as
1434	3	a = Host
31	100	b = NAT

After that, I applied J48 in Weka to the same dataset, which is also known as C4.5. J48 produces a pruned decision tree:

```
number of unique contacted IP addresses <= 6: Host (6997.0/80.0)
number of unique contacted IP addresses > 6
|   number of unique contacted IP <= 11
|   |   number of unique contacted IP <= 9: Host (282.0/108.0)
|   |   number of unique contacted IP > 9: NAT (119.0/37.0)
|   number of unique contacted IP > 11
|   |   number of unique contacted IP <= 35: NAT (418.0/1.0)
|   |   number of unique contacted IP > 35
|   |   |   number of unique user-agents <= 0: NAT (6.0)
|   |   |   number of unique user-agents > 0: Host (18.0/7.0)
```

The size of the decision tree is 11, and there were 6 leaves in the tree. Only 2 out of 8 attributes in the dataset were used to build the decision tree: number of unique contacted IP addresses, and number of unique user-agents. The marked attributes were used when building the tree, unmarked attributes were not used. The selected attributes are the number of unique contacted IP addresses and the number of unique user-agents. Other attributes are considered to have less impact on classification results. In Komárek's work, all the eight attributes were used in the SVM classifier, although he didn't mention how the attributes affect the results or if they affect the results or not. If only some attributes make a difference in classifying NAT devices and end hosts, I can use the minimum number of attributes to detect NAT devices.

Classification results of J48(C4.5) are shown in Table 4.3. In total, 1533 out of 1568 instances were classified correctly, and the accuracy is almost the same with that of SVM, which is 97.77% compared with 97.83% of SVM. J48 correctly classifies only one instance fewer than SVM does. Table 4.4 shows the confusion matrix of J48 results. This confusion matrix is also quite similar to that of SVM, only one more class end host is classified falsely as class NAT device. The TPR of class end host was 1433/1437, 99.72%, and the TPR of class NAT device was 100/131, 76.34%. Similar to SVM results, the TPR of class NAT device was about 20% lower than that of class end host.

Table 4.3 J48 results using 20% as test dataset

		Proportion
Correctly Classified Instances	1533	97.77%
Incorrectly Classified Instances	35	2.23%
Kappa statistic	0.8392	
Total Cost	35	
Average Cost	0.0223	
Mean absolute error	0.0392	
Root mean squared error	0.1351	
Relative absolute error	24.63%	
Root relative squared error	48.79%	
Total Number of Instances	1568	

Table 4.4 J48 confusion matrix

Confusion Matrix		
a	b	<-- classifies as
1433	4	a = Host
31	100	b = NAT

The last machine learning algorithm I applied on the datasets from Aug 02 was the Alternating decision tree (AD Tree). Weka produces the decision tree below:

```
| (1)number of unique contacted IP addresses < 4.5: -1.454
| | (2)number of unique contacted IP addresses < 2.5: -3.002
| | (2)number of unique contacted IP addresses >= 2.5: 0.902
| | (8)number of unique contacted IP addresses < 3.5: -0.219
| | (8)number of unique contacted IP addresses >= 3.5: 0.206
| (1)number of unique contacted IP addresses >= 4.5: 1.252
| | (3)number of unique contacted IP addresses < 10.5: -0.599
| | | (4)number of unique contacted IP addresses < 6.5: -0.481
| | | (4)number of unique contacted IP addresses >= 6.5: 0.352
| | | | (9)number of unique contacted IP addresses < 7.5: -0.262
| | | | (9)number of unique contacted IP addresses >= 7.5: 0.142
| | (3)number of unique contacted IP addresses >= 10.5: 1.328
| | | (5)number of persistent connections < 11: 0.298
| | | | (6)number of unique contacted IP addresses < 11.5: -1.159
| | | | (6)number of unique contacted IP addresses >= 11.5: 3.304
```

```
| | | (5)number of persistent connections >= 11: -1.467
| (7)number of unique contacted IP addresses < 2.5: -1.524
| (7)number of unique contacted IP addresses >= 2.5: 0.003
Legend: -ve = Host, +ve = NAT
```

The total number of nodes in the decision tree was 28, and the number of leaves was 19. Still, only two attributes were used to build the decision tree, as is shown in Table 4.4. Different from J48, the AD Tree uses the attribute number of persistent connections instead of the attribute number of unique user-agents to build the tree. The attribute number of unique contacted IP addresses was used in both decision trees. That is, the other five attributes: number of unique OS and versions, number of unique browser and versions, number of upload bytes, number of download bytes and number of sent HTTP requests are considered to have no influence on classification for both algorithms. The attribute number of persistent connections reflects the amount of traffic, while the attribute number of unique user-agents might reflect the number of systems in one host, it can be a NAT device contains several end hosts or simply be multiple systems on one host. These two attributes affect NAT detection from different aspects. From the point of accuracy, the AD Tree classifier performs better than the J48 classifier.

Table 4.5 Attributes used to build AD Tree decision tree

Attributes in dataset	Used in decision tree
number of unique contacted IP addresses	v
number of unique user-agents	
number of unique OS and versions	
number of unique browser and versions	
number of persistent connections	v
number of upload bytes	
number of download bytes	
number of sent HTTP requests	

Classification results of AD Tree are shown in Table 4.6. In total, 1533 out of 1568 instances were classified correctly. And the accuracy was exactly the same with that of J48, 97.77%. Only 2.23% (35 instances) were incorrectly classified. Apart from the accuracy, the experiment

on the AD Tree had the same confusion matrix as J48, as is shown in Table 4.7. These two decision trees used different algorithms with one common attribute and one different attribute but produced the same accuracy and the same TPR for both classes. Compared with SVM algorithm, decision trees used fewer attributes, but had almost exactly the same accuracy. From these three experiments, either decision tree algorithm can be used for quick NAT detection. When extracting attributes from packets, only a quarter of the eight attributes used in Komárek's work are necessary. But either the training dataset or the test dataset was labelled by generating artificial NATs mentioned in section 3.2.2. To validate the NAT detection methodology, experiments on datasets with real NATs are required.

Table 4.6 AD Tree results using 20% as test dataset

		Proportion
Correctly Classified Instances	1533	97.77%
Incorrectly Classified Instances	35	2.23%
Kappa statistic	0.8392	
Total Cost	35	
Average Cost	0.0223	
Mean absolute error	0.0612	
Root mean squared error	0.1468	
Relative absolute error	38.44%	
Root relative squared error	53.02%	
Total Number of Instances	1568	

Table 4.7 AD Tree confusion matrix

Confusion Matrix		
a	b	<-- classifies as
1433	4	a = Host
31	100	b = NAT

4.2.2 Training and validation

Apart from using 20% of the dataset as the test dataset, I also experimented on a validation dataset on Aug 02 as mentioned in section 3.2.3.2. This validation dataset included packets from one real NAT device and 13 end hosts from the lab. Table 4.8 shows SVM results using this validation dataset. In total 13 instances out of 14 were classified correctly, and the accuracy was 92.86%. One instance was classified incorrectly. This accuracy was a little bit lower than that of experiments on the test dataset. Table 4.9 shows the confusion matrix of SVM algorithm using the validation dataset. All the 14 instances are classified as class end host, that is, although all the instances of class end host were classified correctly, the real NAT device was not detected.

Table 4.8 SVM results using validation dataset

		Proportion
Correctly Classified Instances	13	92.86%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	0	
Total Cost	1	
Average Cost	0.0714	
Mean absolute error	0.0714	
Root mean squared error	0.2673	
Relative absolute error	48.25%	
Root relative squared error	103.52%	
Total Number of Instances	14	

Table 4.9 SVM validation confusion matrix

Confusion Matrix		
a	b	<-- classifies as
13	0	a = Host
1	0	b = NAT

After I processed with the SVM classifier, the validation dataset was processed with the J48 classifier in Weka. The results are shown in Table 4.10. In total, 12 out of 14 instances were classified correctly. The accuracy was 85.71%, lower than that of SVM, 92.86%. Two instances were classified incorrectly. Table 4.11 shows the confusion matrix of J48 algorithm using the

validation dataset. Except for one instance of class end host was classified incorrectly, the other 12 instances of this class were classified correctly. But as with the SVM classifier, the real NAT device is still not detected.

Table 4.10 J48 results using validation dataset

		Proportion
Correctly Classified Instances	12	85.71%
Incorrectly Classified Instances	2	14.29%
Kappa statistic	-0.0769	
Total Cost	2	
Average Cost	0.1429	
Mean absolute error	0.3988	
Root mean squared error	0.4232	
Relative absolute error	269.38%	
Root relative squared error	163.93%	
Total Number of Instances	14	

Table 4.11 J48 validation confusion matrix

Confusion Matrix		
a	b	<-- classifies as
12	1	a = Host
1	0	b = NAT

I also applied the AD Tree classifier to the validation dataset in Weka. Table 4.12 shows the results. In total, 13 out of 14 instances were classified correctly, and the accuracy was 92.86%. The accuracy of the AD Tree was the same as that of SVM, a little bit higher than that of the J48 classifier. Only one instance was classified incorrectly. Table 4.13 shows the confusion matrix of AD Tree. It was exactly the same as that of the SVM classifier. All the instances of class end host were classified correctly, and the real NAT device was not detected.

Table 4.12 AD Tree results using validation dataset

		Proportion

Correctly Classified Instances	13	92.86%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	0	
Total Cost	1	
Average Cost	0.0714	
Mean absolute error	0.4569	
Root mean squared error	0.4741	
Relative absolute error	308.61%	
Root relative squared error	183.65%	
Total Number of Instances	14	

Table 4.13 AD Tree validation confusion matrix

Confusion Matrix		
a	b	<-- classifies as
13	0	a = Host
1	0	b = NAT

Over all, results on the test dataset are better than that on the validation dataset. Although the accuracies of experiments on both test dataset and validation dataset were quite high, all of the three classifiers were unable to detect the real NAT device from the validation dataset. Since my experiment is about detecting NAT devices, I looked into the training dataset to look for the reason why the real NAT device was not detected. There were a large number of inactive end hosts in the dataset, and this led to the generated artificial NAT devices becoming inactive. Values of most of the attributes in inactive hosts or NAT devices were zero: this blurred the boundary between end hosts and NAT devices. In addition, an inactive NAT without end hosts sending packets behind it is meaningless for further host identification experiment. In order to eliminate the effects of inactive NATs, I excluded inactive ones when generating artificial NATs. That is, only artificial NAT with at least half nonzero attributes (4 attributes) were kept for the training dataset. I used this improved artificial NAT generation method and applied the three classifiers on the new training dataset and the previous validation dataset.

Table 4.14 SVM results using validation dataset and improved training dataset

		Proportion
Correctly Classified Instances	13	92.86%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	0	
Mean absolute error	0.0714	
Root mean squared error	0.2673	
Relative absolute error	89.27%	
Root relative squared error	100.94%	
Total Number of Instances	14	

Table 4.15 SVM confusion matrix using validation dataset and improved training dataset

Confusion Matrix		
a	b	<-- classifies as
13	0	a = Host
1	0	b = NAT

Table 4.14 shows the SVM results using the validation dataset and the improved training dataset. And Table 4.15 shows the corresponding confusion matrix. This improved training dataset generated the exact same results as the previous training dataset. That is, the SVM classifier was resistant to those inactive NAT instances. Then the J48 classifier were applied to the new training dataset and the validation dataset. Table 4.16 shows the results using the validation dataset and the improved training dataset on the J48 classifier, and Table 4.17 shows the corresponding confusion matrix. Compared with results on the previous training dataset, all the end hosts were classified correctly, accuracy was improved from 85.71% to 92.86%. But the real NAT device was still undetected.

Table 4.16 J48 results using validation dataset and improved training dataset

		Proportion
Correctly Classified Instances	13	92.86%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	0	

Mean absolute error	0.0788	
Root mean squared error	0.2651	
Relative absolute error	98.49%	
Root relative squared error	100.12%	
Total Number of Instances	14	

Table 4.17 J48 confusion matrix using validation dataset and improved training dataset

Confusion Matrix		
a	b	<-- classifies as
13	0	a = Host
1	0	b = NAT

At last, the AD Tree classifier was applied on the improved training dataset. The results are shown in Table 4.18, and the corresponding confusion matrix is shown in Table 4.19. The accuracy remained the same compared with that in the experiment using the previous training dataset, but the real NAT was detected. The produced decision tree used the attribute number of unique contacted IP addresses. This also support for the idea that not all the attributes provided in Koma´rek's work make a difference on NAT detection. According to the experiments above, the attributes of use are number of unique contacted IP addresses, number of persistent connections, and number of unique user-agents. As the first classifier of being able to detect the real NAT device in the validation dataset, AD Tree is sensitive to inactive NAT instances in the training dataset. Although the three classifiers provided the same accuracy on classifying end hosts and NAT devices, AD Tree stood out from the other classifiers in that it successfully detected the real NAT device.

Table 4.18 AD Tree results using validation dataset and improved training dataset

		Proportion
Correctly Classified Instances	13	92.86%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	0.6316	
Mean absolute error	0.2559	
Root mean squared error	0.315	
Relative absolute error	319.87%	

Root relative squared error	118.98%	
Total Number of Instances	14	

Table 4.19 AD Tree confusion matrix using validation dataset and improved training dataset

Confusion Matrix		
a	b	<-- classifies as
12	1	a = Host
0	1	b = NAT

Figure 4.1 shows the comparison of NAT detection accuracy in the experiments on the test dataset, the validation dataset and the validation dataset with improved training dataset using different machine learning classifiers. Over all, the accuracy was around 90%, and the accuracy on the test dataset was higher than that on the validation dataset. The accuracy on the validation dataset in the experiments with the original training dataset or the improved training dataset was almost the same, except for accuracy in the experiment using J48 improved a little when it was trained on the improved training dataset. Since the AD Tree classifier detected the real NAT device successfully with the improved training dataset, I suggest AD Tree as the classifier for NAT detection instead of the other two classifiers.

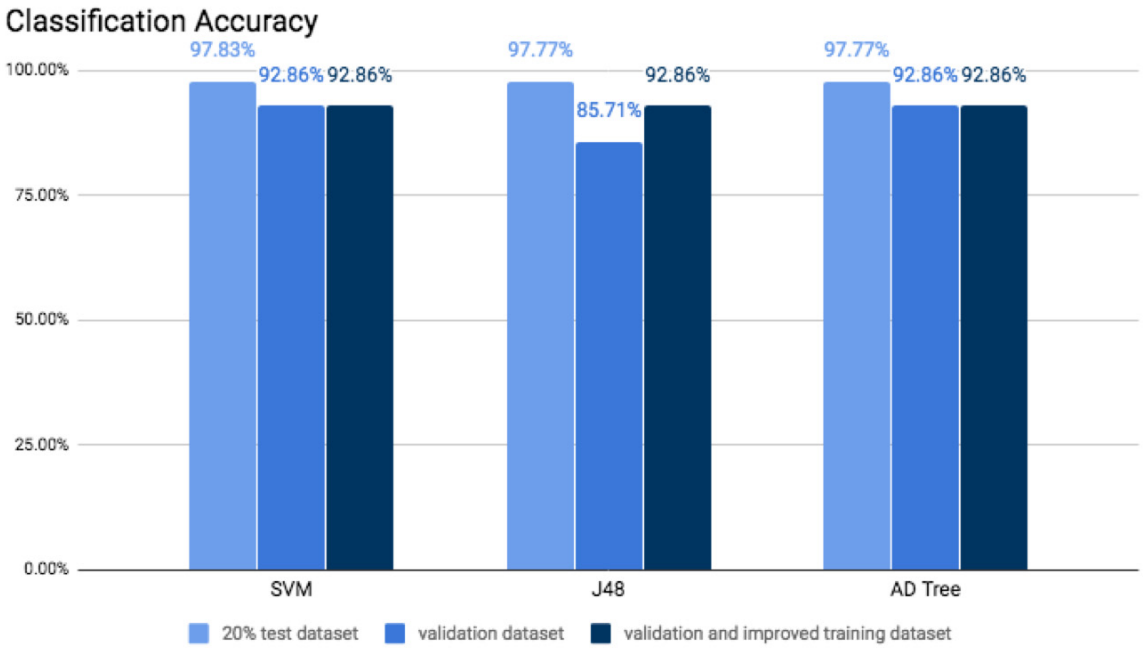

Figure 4.1 Classification accuracy

4.2.3 NAT detection on more datasets using AD Tree

Since the AD Tree classifier performed best in the NAT detection experiments above, It was then applied on the other datasets captured on four different dates. The datasets used were captured on Aug 04 (with all the hosts running), Aug 14 (with only Kali hosts running), Aug 15 (with only Windows hosts running) and Aug 17 (with half of the Windows hosts and half of the Kali hosts running). I first performed experiments on a training dataset and a validation dataset from a single date as in the experiments in Section 4.2.2. Then I conducted experiments using a training dataset from one date and a validation dataset from another date in order to study the generalization of the NAT detection approach in the research.

Table 4.20 shows the AD Tree classification results on the training dataset and the validation dataset from Aug 04, and Table 4.21 shows the corresponding confusion matrix. The AD tree classifier achieved a good result with 100% accuracy. All the end hosts and the NAT device were classified correctly. In addition to the attribute number of unique contacted IP addresses used in the previous NAT detection experiments, the attribute number of download bytes was also used in building AD Tree for this training dataset. The additional attribute number of download bytes in this experiment might be the reason for the 100% accuracy compared with the 92.86% accuracy of AD classification on the dataset captured on Aug 02.

Table 4.20 AD Tree results using training dataset and validation dataset on Aug 04

		Proportion
Correctly Classified Instances	19	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.051	
Root mean squared error	0.0678	
Relative absolute error	95.82%	
Root relative squared error	29.56%	
Total Number of Instances	19	

Table 4.21 AD Tree confusion matrix using training dataset and validation dataset on Aug 04

Confusion Matrix		
a	b	<-- classifies as
18	0	a = Host
0	1	b = NAT

The next experiment was conducted on the training dataset and the validation dataset captured on Aug 14 with only Kali hosts running. The decision tree generated by AD classifier on this training dataset contained one more attribute than the tree generated on dataset captured on Aug 04 had (number of download bytes, and number of unique contacted IP addresses): number of persistent connections. Table 4.22 shows the AD Tree classification results, and Table 4.23 shows the corresponding confusion matrix. AD Tree classified all the instances correctly, including end hosts and the NAT device. The accuracy remained 100% as that in the previous experiment.

Table 4.22 AD Tree results using training dataset and validation dataset on Aug14 (Kali hosts only)

		Proportion
Correctly Classified Instances	8	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.0765	
Root mean squared error	0.1588	
Relative absolute error	60.93%	
Root relative squared error	44.94%	
Total Number of Instances	8	

Table 4.23 AD Tree confusion matrix using training dataset and validation dataset on Aug14 (Kali hosts only)

Confusion Matrix		
a	b	<-- classifies as
7	0	a = Host
0	1	b = NAT

Then datasets from Aug 15 with only Windows hosts running were employed in the AD Tree classifier. The decision tree generated on this training dataset used three attributes, including: number of unique contacted IP addresses, number of unique user-agents, and number of upload bytes. Table 4.24 shows the results, and Table 4.25 shows the corresponding confusion matrix. Although using two different attributes in the decision tree, the accuracy of the AD Tree classification experiment on datasets captured on Aug 15 also remained 100%.

Table 4.24 AD Tree results using training dataset and validation dataset on Aug 15 (Windows hosts only)

		Proportion
Correctly Classified Instances	9	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.1218	
Root mean squared error	0.2078	
Relative absolute error	108.60%	
Root relative squared error	62.42%	
Total Number of Instances	9	

Table 4.25 AD Tree confusion matrix using training dataset and validation dataset on Aug 15 (Windows hosts only)

Confusion Matrix		
a	b	<-- classifies as

8	0	a = Host
0	1	b = NAT

The results of the last NAT detection experiment on the training dataset and the validation dataset captured on the same date are shown in Table 4.26. It used datasets from Aug 17 with half of the Kali hosts and half of the Windows hosts running. The generated decision tree by the AD Tree classifier generated used three attributes: number of unique contacted IP addresses, number of upload bytes and number of download bytes. The AD Tree classifier still classified end hosts and the NAT device correctly, with an accuracy of 100%. The confusion matrix is shown in Table 4.27.

Table 4.26 AD Tree results using training dataset and validation dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running)

		Proportion
Correctly Classified Instances	9	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.1306	
Root mean squared error	0.1882	
Relative absolute error	116.71%	
Root relative squared error	56.52%	
Total Number of Instances	9	

Table 4.27 AD Tree confusion matrix using training dataset and validation dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running)

Confusion Matrix		
a	b	<-- classifies as
8	0	a = Host
0	1	b = NAT

The above experiments used a training dataset and a validation dataset captured from different interfaces on the same date. After those experiments, I performed NAT detection experiments

on a training dataset and a validation dataset captured on different dates with different hosts running in order to study the impact of different operating systems on NAT detection and to test the generalization of this approach.

Table 4.28 shows the results of AD Tree classification using the training dataset on Aug 02 and the validation dataset on Aug 04. The decision tree generated contained only one attribute: number of unique contacted IP addresses. The datasets of both dates were collected with all the hosts running on the LAN. The NAT detection experiment on the training dataset and the validation dataset from different dates performed worse than that on the training dataset and the validation dataset from the same date, with an accuracy of 84%. The confusion matrix of the classification results is shown in Table 4.29. The real NAT was detected, while three end hosts were incorrectly classified as NAT devices. Different amounts of traffic in the training datasets captured on different dates led to different behavioral models of NAT devices. Thus, the end host instances with a large amount of traffic might be considered to have features of NAT devices by the decision tree classifier. In addition, only one attribute was used in generated decision tree might be another reason for the incorrect classification. Over all, this NAT detection approach provides fairly accurate classification on the training dataset and the validation dataset captured on different dates.

Table 4.28 AD Tree results using training dataset on Aug 02 and validation dataset on Aug 04

		Proportion
Correctly Classified Instances	16	84%
Incorrectly Classified Instances	3	16%
Kappa statistic	0.3448	
Mean absolute error	0.2766	
Root mean squared error	0.3084	
Relative absolute error	449.12%	
Root relative squared error	135.68%	
Total Number of Instances	19	

Table 4.29 AD Tree confusion matrix using training dataset on Aug 02 and validation dataset on Aug 04

Confusion Matrix		
a	b	<-- classifies as
15	3	a = Host
0	1	b = NAT

The previous experiment was performed on the training dataset from Aug 02 and the validation dataset from Aug 04. Table 4.30 shows the NAT detection results on the training dataset from Aug 04 and the validation dataset from Aug 02, and Table 4.31 shows the corresponding confusion matrix. The generated decision tree contained two attributes: number of download bytes, and number of unique contacted IP addresses. The accuracy was relatively low: 57%. The NAT device was detected; however, six out of 13 end hosts were incorrectly classified as NAT devices.

Table 4.30 AD Tree results using training dataset on Aug 04 and validation dataset on Aug 02

		Proportion
Correctly Classified Instances	8	57%
Incorrectly Classified Instances	6	43%
Kappa statistic	0.1429	
Mean absolute error	0.3456	
Root mean squared error	0.4878	
Relative absolute error	479.67%	
Root relative squared error	182.65%	
Total Number of Instances	14	

Table 4.31 AD Tree confusion matrix using training dataset on Aug 04 and validation dataset on Aug 02

Confusion Matrix		
a	b	<-- classifies as
7	6	a = Host
0	1	b = NAT

The above two experiments were performed on a training dataset and a validation dataset from different dates, but both of the datasets were with all the end hosts running. The following experiment used the training dataset with half of the hosts running on one day and the validation dataset with the other hosts running on another day. Table 4.32 shows the results on the training dataset from Aug 14 with only Kali hosts running and on the validation dataset from Aug 15 with only Windows hosts running. The generated decision tree contained three attributes: the number of download bytes, number of unique contacted IP addresses, and number of persistent connections. The accuracy was unexpectedly high: 100%. Both the instance of class NAT and the instances of class end host were classified correctly. The behavioral model built from the dataset with only Kali hosts running perfectly classified the target dataset with only Windows hosts running.

Table 4.32 AD Tree results using training dataset on Aug 14 (Kali hosts only) and validation dataset on Aug 15 (Windows hosts only)

		Proportion
Correctly Classified Instances	9	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.0619	
Root mean squared error	0.1342	
Relative absolute error	55.49%	
Root relative squared error	40.29%	
Total Number of Instances	9	

Table 4.33 AD Tree confusion matrix using training dataset on Aug 14 (Kali hosts only) and validation dataset on Aug 15 (Windows hosts only)

Confusion Matrix		
a	b	<-- classifies as
8	0	a = Host
0	1	b = NAT

Then I used the dataset captured on Aug 15 with only Windows hosts running as the training dataset and the dataset captured on Aug 14 with only Kali hosts running as the validation dataset. The results are shown in Table 4.34, and corresponding confusion matrix is shown in Table 4.35. The generated decision tree contained three attributes: number of unique contacted IP addresses, number of unique user-agents, and number of upload bytes. The accuracy was as little as 25%, with the NAT device and only one out of the seven end host instances classified correctly. More than 85% of the instances from class end host were classified incorrectly. This shows that the behavioral model built from the dataset with only Windows hosts running cannot correctly classify the dataset with only Kali hosts running. The most probable cause is that Windows hosts generated less traffic than Kali hosts did in one day due to the different performance of automatic URL generation program on different operating systems, as is described in Section 3.1.2.

Table 4.34 AD Tree results using training dataset on Aug 15 (Windows hosts only) and validation dataset on Aug 14 (Kali hosts only)

		Proportion
Correctly Classified Instances	2	25%
Incorrectly Classified Instances	6	75%
Kappa statistic	0.04	
Mean absolute error	0.6528	
Root mean squared error	0.7253	
Relative absolute error	517.92%	

Root relative squared error	205.43%	
Total Number of Instances	8	

Table 4.35 AD Tree confusion matrix using training dataset on Aug 15 (Windows hosts only) and validation dataset on Aug14 (Kali hosts only)

Confusion Matrix		
a	b	<-- classifies as
1	6	a = Host
0	1	b = NAT

The next NAT detection experiment used the dataset captured on Aug 17 with half of the Kali hosts and half of the Windows hosts running as the training dataset and the dataset captured on Aug 02 with all the hosts running as the validation dataset. The results are shown in Table 4.36, and corresponding confusion matrix is shown in Table 4.37. The decision tree generated by the AD Tree classifier used three attributes: number of unique contacted IP addresses, number of upload bytes, and number of download bytes. The decision tree built on the training dataset with half of the hosts running correctly classified the NAT device as well as 11 out of 13 end hosts in the validation dataset with all the hosts running, with an accuracy of 86%. Only two end hosts in the validation dataset were classified incorrectly as NAT devices.

Table 4.36 AD Tree results using training dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running) and validation dataset on Aug 02 (all the hosts running)

		Proportion
Correctly Classified Instances	12	86%
Incorrectly Classified Instances	2	14%
Kappa statistic	0.44	
Mean absolute error	0.157	
Root mean squared error	0.2741	
Relative absolute error	217.18%	
Root relative squared error	102.68%	

Total Number of Instances	14	

Table 4.37 AD Tree confusion matrix using training dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running) and validation dataset on Aug 02 (all the hosts running)

Confusion Matrix		
a	b	<-- classifies as
11	2	a = Host
0	1	b = NAT

The following NAT detection experiment used the dataset captured on Aug 02 with all the hosts running as the training dataset and the dataset captured on Aug 17 with half of the Kali hosts and half of the Windows hosts running as the validation dataset. The results are shown in Table 4.38, and corresponding confusion matrix is shown in Table 4.39. The decision tree generated by the AD Tree classifier used a single attribute: number of unique contacted IP addresses. It correctly classified the NAT device and all the end hosts in the validation dataset with half of the hosts running, with an accuracy of 100%.

Table 4.38 AD Tree results using training dataset on Aug 02 (all the hosts running) and validation dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running)

		Proportion
Correctly Classified Instances	9	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.239	
Root mean squared error	0.2468	
Relative absolute error	201.04%	
Root relative squared error	74.75%	
Total Number of Instances	9	

Table 4.39 AD Tree confusion matrix using training dataset on Aug 02 (all the hosts running) and validation dataset on Aug 17 (half of the Kali hosts and half of the Windows hosts running)

Confusion Matrix		
a	b	<-- classifies as
8	0	a = Host
0	1	b = NAT

As summarized in the above experiments, NAT detection provides fairly good results when the training dataset and the validation dataset were collected during the same period. However, its accuracy varies widely from 25% to 100% on datasets captured on different dates. The accuracy and attributes used in the decision are shown in Table 4.40. Among the eight attributes provided in the datasets, five of them were used. Of the other three unused attributes, the number of unique OSs and versions and the number of unique browsers and versions are actually indicated in the attribute number of unique user-agents; and the number of sent HTTP requests reveals the amount of traffic sent from a host, which is similar to the attribute number of persistent connections. It can be concluded from the experimental results that these three attributes have no or little impact on NAT detection. To effectively detect NAT devices, the five attributes listed in Table 4.40 are required. The number of unique contacted IP addresses was adopted by the AD tree classifier in every experiment, and the number of upload bytes and the number of download bytes were used several times. In addition, it can be analyzed in Table 4.40 that the number of attributes and the kinds of attributes used in the decision tree do not necessarily have an effect upon NAT detection accuracy.

Table 4.40 NAT detection accuracies and decision tree attributes on different training datasets and validation datasets

No.	Training dataset	Validation dataset	Attributes used in the decision tree					Accuracy
			Number of unique contacted IP addresses	Number of unique user-agents	Number of persistent connections	Number of upload bytes	Number of download bytes	
1	Aug 02-all the host	Aug 04-all the hosts	v					84%
2	Aug 04-all the hosts	Aug 02-all the hosts	v				v	57%

3	Aug 14-Kali hosts only	Aug 15-Windows hosts only	v		v		v	100%
4	Aug 15-Windows hosts only	Aug 14-Kali hosts only	v	v		v		25%
5	Aug17-half of the hosts	Aug 02-all the hosts	v			v	v	86%
6	Aug 02-all the hosts	Aug17-half of the hosts	v					100%

As is shown in Table 4.40, one of the experiments (No.4) with a low accuracy used the training dataset with only Windows hosts running and the validation dataset with only Kali hosts running, but another (No.2) used the training dataset captured on Aug 02 and the validation dataset captured on Aug 04. Both of the datasets were collected when all the hosts were running. Hence the hosts' operating systems of the datasets do not necessarily have an effect upon the accuracy. The reason for the variation of the accuracy lies in data sizes, as is shown in Table 4.41. The common ground of the two experiments is that they both used a training dataset and a validation dataset that were processed from the original datasets with a large difference in data size. This does not necessarily lead to a low accuracy because the accuracies of No.1 and No.3 remained high despite using the datasets processed from the datasets with a large difference in data size. Hence the disparity between datasets' sizes can lead to uncertain accuracy. The other two experiments, No.5 and No.6, had a good performance on accuracy using datasets with similar data sizes to be processed for the training dataset and the validation dataset. To draw the conclusion that the accuracy of NAT detection is steadily high when it uses a training dataset and a validation dataset that are processed from datasets of similar sizes, more experiments are needed.

Table 4.41 Data size of datasets used in NAT detection experiments

Date	Total data size (GB)
Aug 02-all the host	44
Aug 04-all the hosts	110
Aug 14-Kali hosts only	42
Aug 15-Windows hosts only	13
Aug17-half of the hosts	32

To verify the above theory, I chose datasets captured on Aug 02 (44 GB) and datasets captured on Aug 14 (42 GB) for NAT detection. The datasets from these two dates had the similar sizes, and consequently the NAT detection accuracy using one of the datasets as the training dataset and another as the validation dataset should be steadily high. Results using the training dataset captured on Aug 02 and the validation dataset captured on Aug 14 are shown in Table 4.42, with an accuracy of 100%. Results of using the training dataset captured on Aug 14 and the validation dataset captured on Aug 02 are shown in Table 4.43, with an accuracy of 93%. These two additional NAT detection experiments along with No.5 and No.6 in Table 4.40 verify the finding that the accuracy of NAT detection stays high when the training dataset and the validation dataset are processed from datasets of similar sizes. To detect NAT devices on the network, selecting a training dataset of the similar size to the target dataset contributes to a high accuracy.

Table 4.42 AD Tree results using training dataset on Aug 02 (all hosts running) and validation dataset on Aug 14 (Kali hosts only)

		Proportion
Correctly Classified Instances	8	100%
Incorrectly Classified Instances	0	0%
Kappa statistic	1	
Mean absolute error	0.2417	
Root mean squared error	0.2502	
Relative absolute error	182.43%	
Root relative squared error	71.46%	
Total Number of Instances	8	

Table 4.43 AD Tree results using training dataset on Aug 14 (Kali hosts only) and validation dataset on Aug 02 (all hosts running)

		Proportion
Correctly Classified Instances	13	93%
Incorrectly Classified Instances	1	7.14%
Kappa statistic	1	

Mean absolute error	0.6316	
Root mean squared error	0.0929	
Relative absolute error	128.95%	
Root relative squared error	82.19%	
Total Number of Instances	14	

4.3 Host identification experiments

Our experiment in this phase was implemented with the Python programming language. Datasets with packets separated by connection were processed with Tshark as presented in Section 3.3. This software then processed the dataset through the linear regression algorithm, and then compared the lines generated to identify hosts as mentioned above. We applied the linear-model package from Scikit-learn (sklearn) to implement least squares linear regression. sklearn is a free software ML library provided for the Python language. Its dependencies are NumPy and SciPy, and they are also Python libraries.

4.3.1 Evaluation criteria

Each connection from the dataset was marked with a pair of (a,su_0), where a is the slope and su_0 is the x value of the point when the generated line crosses the x axis. Connections with same sets of (a,su_0), or similar enough sets (method of comparing distance with threshold in section 3.3) were considered to belong to the same host. From datasets captured on the LAN interface, packets passed through the NAT device were matched by connection with original packets directly sent from the end hosts. That is, host identification determines if several connections are associated with the same end host or not. In this way, evaluation criteria work in two ways. The first method evaluates whether the connections considered to belong to the same host by the program are really sent from the same host (evaluation method A). The second method evaluates whether original connections from the same host are identified to belong to the same hosts (might be several hosts) by the program (evaluation method B). These two evaluation methods provide different benchmarks from different angles for my experiment.

They provide very different results. In my experiment, I applied these two evaluation methods for host identification.

4.3.2 Host identification experiments and results

According to the TCP timestamp definition in RFC 1323 [6], the TCP timestamp value is at least approximately proportional to real time. Hence the increment rate of timestamp, or the slope should be positive in the research. No matter what the exact value of the slope is, a negative slope is an incorrect result.

When re-engineering Georg Wicherski's experiments, some computed slopes were negative. To evaluate the parameters impacting the experiment results, I first conducted experiments with a variable number of packets in one connection. Only connections with more than or equal to the given number of packets were taken into consideration. Results are shown in Table 4.44. The proportion of the number of hosts with a positive slope increased at first and then decreased with the increase of the number of packets in one connection, peaking at 55 packets in one connection (98.86%). And the proportion of positive slopes of all slopes increased from 50% (30 packets in one connection) to 72.7% (55 packets in one connection). This trend indicates that connections not containing enough packets led to calculation errors when generating the artificial lines of timestamp value and system uptime using least-squares linear regression. Also, this number should not be as large as 100 or 200, because applying too large a number will filter out too many connections. As is shown in Table 4.44, using connections with more than 200 packets, it can only detect 678 hosts in total, compared with 5515 hosts using connections with more than 30 packets. So, in this research, I adopted 55 as the limit for the number of packets in one connection.

Table 4.44 Number of packets in one connection (threshold = 2ms)

number of packets in one connection	number of slopes	negative slope	number of hosts	number of hosts with negative slope	Proportion
30	22	11	5515	20	99.64%
40	18	10	4085	13	99.68%

50	14	5	3210	6	99.81%
55	11	3	2901	4	99.86%
60	9	3	2664	4	99.85%
100	6	3	1547	3	99.81%
200	2	1	678	2	99.71%

The following experiment filtered out connections with fewer than 55 packets and tested on different threshold values $\delta_{\square\square\square\square}$ (limit values to decide if two lines are close enough). Threshold values varied from 2 ms to 600 ms. All the other parameters were settled as in Georg Wicherski's experiments [5]. Results are shown in Table 4.45. Accuracy rose from 13% at threshold value 2 ms to 23% at threshold value 600 ms. In addition, the number of detected hosts with negative slopes (those with errors) decreased from 4 (threshold value 2 ms to 50 ms) to 0 (threshold value 100 ms to 600 ms). Although the accuracy increased to a certain degree with the increase of the threshold value, it was still below expectations. Other experiments were performed to explore the reasons for this and improve the research.

Table 4.45 Results on original method

Threshold(ms)/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	11	10	9	8	6	5	3	2	2
Number of hosts with negative slopes	4	4	4	4	4	0	0	0	0
Number of hosts	3110	2999	2872	2691	2341	1949	1568	1247	1089
Number of negative slopes	3	2	2	2	1	0	0	0	0
Accuracy	13%	14%	14%	15%	16%	18%	19%	22%	23%

After experimenting on the number of packets in one connection and different threshold values, I adopted an improved calculation method for the distance between two lines and kept the other parameters the same. Experimental results are shown in Table 4.46. The change in the number of negative slopes and the number of hosts with negative slopes were the same as those of the previous experiment. Accuracy increased to a greater extend compared with that in the experiment using the original distance calculation method, from 14% at threshold value 2 ms

to 36% at threshold value 600 ms. The new distance method improved the accuracy more when the threshold value was larger.

Table 4.46 Results on improved distance between two lines calculation method

Threshold(ms)/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	11	10	9	8	6	5	3	2	2
Number of hosts with negative slopes	4	4	4	4	4	0	0	0	0
Number of hosts	2902	2679	2399	2060	1554	1237	965	731	639
Number of negative slopes	3	2	2	2	1	0	0	0	0
Accuracy	14%	15%	16%	16%	19%	22%	29%	34%	36%

The following experiment had the same parameters as the previous one but adopted an improved method for flow separation. The results are shown in Table 4.47. This method eliminated negative slopes no matter the threshold value. The accuracy in this experiment also improved distinctly on the whole, the accuracy varied from 34% at threshold value 2 ms (compared with the lowest accuracy 14% in Table 4.46) to 42% at threshold value 100 ms (compared with the highest accuracy 36% in Table 4.46). The accuracy peaked at the threshold value 100 ms and fluctuated around 40% as the threshold value increased. From both sides, the new flow separation method contributed significantly to host identification.

Table 4.47 Results on improved flow separation method and improved distance calculation method

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	90	88	83	80	72	65	52	43	40
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	34%	34%	38%	38%	39%	42%	35%	40%	35%

The last experiment differed from the others in its evaluation criteria. It used the evaluation method B mentioned in Section 4.3.1 to judge if original connections from the same host were identified as belonging to the same host. Results are shown in Table 4.48. All the parameters remained the same as those in the experiment in Table 4.47.

Table 4.48 Results on improved flow separation method and improved distance calculation method and new evaluation method

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	90	88	83	80	72	65	52	43	40
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

The most obvious improvement was that the accuracy was 100% regardless of the value of threshold. That is, even though connections sent from one host were detected as sent from several different end hosts, as the results of evaluation method A in Table 4.47 reveal, results under the evaluation method B reflect that connections detected as from the same host were indeed sent from the same host.

Table 4.49 Host identification accuracy comparison with different parameters

threshold/number of packets in one connection			2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Distance	Connection separation	Evaluation									
paper	paper	host -> connection (A)	13%	14%	14%	15%	16%	18%	19%	22%	23%
proportional	paper	host -> connection (A)	14%	15%	16%	16%	19%	22%	29%	34%	36%

proportional	flow	host -> connection (A)	34%	34%	38%	38%	39%	42%	35%	40%	35%
proportional	flow	connection -> host (B)	100%	100%	100%	100%	100%	100%	100%	100%	100%

Accuracy comparisons of all the above experiments are shown in Table 4.49, and the corresponding line chart is shown in Figure 4.2. Parameters labeled as paper are from Georg Wicherski's work [5]. The first three rows results were evaluated by criterion host to connection (Evaluation method A). The accuracy of these three rows were relatively low compared with that of the last row using the evaluation method from connection to host (Evaluation method B). But each improved parameter contributes to a higher accuracy. The improvement of accuracy from the third experiment to the second experiment was greater than the improvement of the accuracy from the second experiment to the first one. That is, the improved flow separation method contributes more to the accuracy than the proportional distance calculation method does. As is shown in Figure 4.2, the accuracy of experiments using the old connection separation method first increased slowly with the increment of threshold value, and then it rose rapidly at threshold value 100 ms to threshold value 200 ms, followed by a mild increase. The accuracy increased first and then fluctuated as the threshold value raised. While the accuracy of the fourth experiment remained very high regardless of different threshold values as discussed in the previous paragraph.

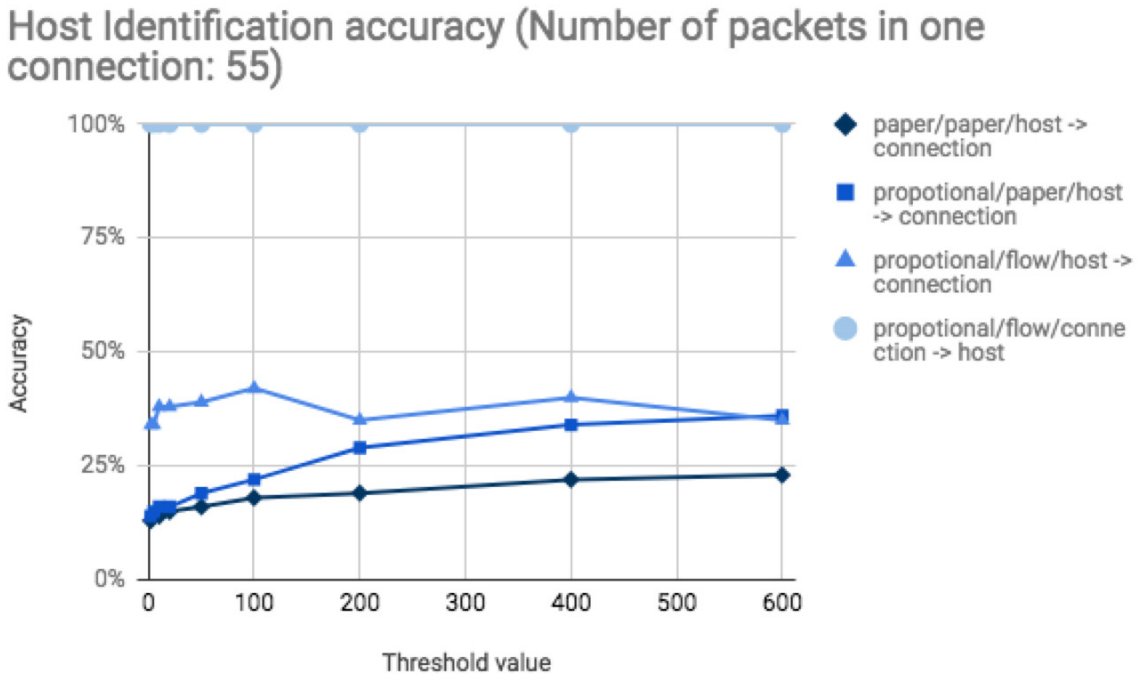

Figure 4.2 Host identification accuracy

4.3.3 Host identification experiments using datasets from NAT detection

The host identification approach was then applied on other datasets captured on different dates: Aug 02, Aug 14, Aug 15, and Aug 17. Each experiment was evaluated by the two evaluation methods mentioned above. Table 4.50 shows the results on the dataset captured on Aug 02 using evaluation method A. The value of number of packets in one connection used in this and the following experiments was 55. Hosts of both operating systems were detected. Since only one slope$a$ was detected regardless of different threshold values, and according to the fact that a's are determined by operating systems, the difference of the TCP timestamp values' increment rates a's between the two operating systems was smaller than any threshold value used in this research. In other words, the increment rates of TCP timestamp values on Kali hosts and Windows hosts can be considered as the same in this research. The number of detected hosts decreased from 41 at the threshold value 2 ms to 26 at the threshold value 600 ms. It was much larger than the number of actual hosts (16) regardless of different threshold values. The accuracy increased slowly from 65% to 74% as the threshold value increased. The accuracy of 65% at threshold value 2 ms means that on average 65% connections sent from the same host were identified to belong to the same host by the program.

Table 4.50 Host identification results using evaluation method A on dataset from Aug 02

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	41	41	41	39	37	35	31	28	26
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	65%	65%	65%	65%	65%	65%	65%	72%	74%

The results on the same dataset using evaluation method B are shown in Table 4.51. The results of detected hosts and slopes were the same as in the experiment using evaluation method A. The host identification accuracy maintained 100% regardless of different threshold values. This reveals that all the original connections sent from the same host were identified to belong to same hosts (can be more than one host) by the program. Although the program detected more hosts than the number of actual end hosts behind the NAT device, connections of any detected host were indeed sent from the same end host according to the dataset collected on the internal interface on the LAN.

Table 4.51 Host identification results using evaluation method B on dataset from Aug 02

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	41	41	41	39	37	35	31	28	26
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.52 shows the host identification results on the dataset captured on Aug 14 with only Kali hosts running using evaluation method A. Only one a was detected, which was still consistent with the principle that a's are determined by operating systems. The number of detected hosts decreased from 24 at the threshold value 2 ms to 10 at the threshold value 600 ms. The number of detected hosts at the threshold value 2 ms tripled the actual number of hosts running on this date (eight). The accuracy increased obviously as the threshold value increased.

Table 4.52 Host identification results using evaluation method A on dataset from Aug 14

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0

Number of hosts	24	24	22	21	19	17	14	10	10
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	61%	62%	62%	62%	70%	87%	88%	88%	89%

Host identification results on the dataset captured on Aug 14 with only Kali hosts running using evaluation method B are shown in Table 4.53. The accuracy remained 100% regardless of different threshold values. This still indicates that connections of any detected host were actually sent from the same end host.

Table 4.53 Host identification results using evaluation method B on dataset from Aug 14

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	24	24	22	21	19	17	14	10	10
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Host identification results on another dataset captured on Aug 15 with only Windows hosts running using evaluation method A are shown in Table 4.54. Only one a was detected on this dataset as in the previous experiment using evaluation method A. The number of detected hosts decreased from 11 at the threshold value 2 ms to 9 at the threshold value 600 ms. The number of detected hosts at the threshold value 2 ms was quite similar to the actual number of hosts running on that date (eight). The accuracy increased more slowly as the threshold value increased compared with that in the previous experiment, and it maintained 62% while the threshold value increased from 2 ms to 400 ms.

Table 4.54 Host identification results using evaluation method A on dataset from Aug 15

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55

Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	11	11	11	11	10	10	10	10	9
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	62%	62%	62%	62%	62%	62%	62%	62%	65%

Table 4.55 shows results on the dataset captured on Aug 15 using evaluation method B. The accuracy still maintained 100% regardless of the threshold values. The program detected a few more hosts than the number of actual end hosts behind the NAT device, but still, connections of any detected host were actually sent from the same end host in the dataset captured from the internal interface.

Table 4.55 Host identification results using evaluation method B on dataset from Aug 15

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	11	11	11	11	10	10	10	10	9
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

The results of the next host identification experiment on another dataset captured on Aug 17 with half of the Windows hosts and half of the Kali hosts running using evaluation method A are shown in Table 4.56. One a is detected, and the number of detected hosts decreased from 64 to 36 when the threshold value increased from 2 ms to 600 ms. The program detected much more hosts than the actual number of end hosts (eight) behind the NAT device. The accuracy increased obviously from 45% to 74% as the threshold value raised.

Table 4.56 Host identification results using evaluation method A on dataset from Aug 17

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	64	64	62	60	53	50	44	40	36
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	45%	45%	53%	68%	69%	73%	73%	74%	74%

Table 4.57 shows results on the dataset captured on Aug 17 using evaluation method B. The accuracy still maintained 100% regardless of different threshold values. The program detected much more hosts than the number of actual end hosts behind the NAT device, but the connections of any detected host were sent from the same end host in the dataset collected on the internal interface.

Table 4.57 Host identification results using evaluation method B on dataset from Aug 17

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	64	64	62	60	53	50	44	40	36
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

4.4 Host identification on datasets from NAT detection

The previous two sections describe NAT detection and host identification experiments separately. This section describes host identification experiments using datasets from the NAT detection stage. Datasets with only HTTP packets were extracted to get TCP attributes like TCP timestamp values for host identification. HTTP packets only account for a small part of

TCP packets: about 2% to 3% on average in the datasets I collected. Hence the number of packets in one connection in these datasets were smaller than that of datasets used in Section 4.3. The limit for the number of packets in one connection I used in section was 10 instead of 55 in the previous section.

Table 4.58 shows the host identification results using evaluation method A on dataset from the NAT detection stage captured on Aug 02. Only one slope a was detected as in the experiment using datasets directly extracted from the original dataset captured on Aug 02. The number of detected hosts was five at threshold value 2 ms, which was far fewer than that in the experiment on the previous dataset: 41. It was also smaller than the actual number of hosts running on that date: 16. The performance on host identification accuracy was unsatisfying. The number of hosts and the accuracy hardly changed as the threshold value increased.

Table 4.58 Host identification results using evaluation method A on Aug 02 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	4
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	27%	27%	27%	27%	27%	27%	27%	27%	40%

The host identification results using evaluation method B on Aug 02 dataset from the NAT detection stage are shown in Table 4.59. Despite of a fewer number of detected hosts and the lower accuracy using evaluation method A compared with those in the experiment on the original dataset, connections of any detected host were actually sent from the same end host.

Table 4.59 Host identification results using evaluation method B on Aug 02 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10

Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	4
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Results of the experiment on the NAT detection dataset from Aug 04 using evaluation method A are shown in Table 4.60. The number of detected slopes a was one. It detected 18 to 21 hosts at different threshold values. It was lower than the actual number of hosts running and the number of detected hosts on the dataset directly processed from the original dataset captured on Aug 04 (40 to 90 hosts). The host identification accuracy (17% to 22%) was lower than that of the experiment using the original dataset (34% to 42%).

Table 4.60 Host identification results using evaluation method A on Aug 04 dataset from the NAT detection

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	21	21	21	21	21	20	19	19	18
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	17%	17%	17%	17%	17%	17%	19%	19%	22%

The host identification results B on the dataset from the NAT detection stage captured on Aug 04 using evaluation method are shown in Table 4.61. The accuracy remained 100%. Still, connections of any detected host were actually sent from the same end host.

Table 4.61 Host identification results using evaluation method B on Aug 04 dataset from the NAT detection

Threshold/Number of packets in one connection	2/55	5/55	10/55	20/55	50/55	100/55	200/55	400/55	600/55
Number of a	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	21	21	21	21	21	20	19	19	18
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Results of the experiment on the NAT detection dataset captured on Aug 14 using evaluation method A are shown in Table 4.62. The number of detected slopes a was one. It detected at most five hosts. It was lower than the actual number of hosts running or the number of detected hosts on the dataset directly processed from the original dataset captured on Aug 14. The host identification accuracy (59% to 74%) was lower than that of the experiment using the original dataset (61% to 89%).

Table 4.62 Host identification results using evaluation method A on Aug 14 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	4	4
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	59%	59%	59%	59%	59%	59%	59%	74%	74%

The host identification results using evaluation method B on Aug 14 dataset from the NAT detection stage are shown in Table 4.63. Similar to the above experiments on the datasets from NAT identification stage using evaluation method B, the accuracy remained 100%. That is, any detected host were actually sent from the same end host.

Table 4.63 Host identification results using evaluation method B on Aug 14 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	4	4
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.64 shows the host identification results on the dataset from NAT detection stage captured on Aug 15 using evaluation method A. It detected one a and five hosts regardless of different threshold values. The host identification accuracy was 49%, lower than that of the experiment on the original dataset from Aug 15 (62% or 65%). The number of detected hosts was five, smaller than the number of actual hosts running: eight.

Table 4.64 Host identification results using evaluation method A on Aug 15 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	5
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	49%	49%	49%	49%	49%	49%	49%	49%	49%

Table 4.65 shows the host identification results on the dataset from NAT detection stage captured on Aug 15 using evaluation method B. The accuracy was 100% as in the above experiments using evaluation method B. Any detected host were actually sent from the same end host.

Table 4.65 Host identification results using evaluation method B on Aug 15 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	5
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 4.66 shows the host identification results on the dataset from NAT detection stage captured on Aug 15 using evaluation method A. One a was detected as in the experiments on other datasets. The number of detected hosts was five regardless of different threshold values. The host identification accuracy was 49%, lower than the accuracy of the experiment on the original dataset captured on Aug 17 (45% to 74%) at most of the threshold values.

Table 4.66 Host identification results using evaluation method A on Aug 17 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	5
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	49%	49%	49%	49%	49%	49%	49%	49%	49%

Table 4.67 shows the host identification results on the dataset from NAT detection stage captured on Aug 17 using evaluation method B. The accuracy was 100% as in the above experiments using evaluation method B. Any detected host were actually sent from the same end host.

Table 4.67 Host identification results using evaluation method B on Aug 17 dataset from the NAT detection

Threshold/Number of packets in one connection	2/10	5/10	10/10	20/10	50/10	100/10	200/10	400/10	600/10
Number of a's	1	1	1	1	1	1	1	1	1
Number of hosts with minus slopes	0	0	0	0	0	0	0	0	0
Number of hosts	5	5	5	5	5	5	5	5	5
Number of minus slopes	0	0	0	0	0	0	0	0	0
Accuracy	100%	100%	100%	100%	100%	100%	100%	100%	100%

4.5 Summary

This chapter describes and evaluates three kinds of experiments I did on NAT detection, host identification, and NAT detection and host identification (host identification using datasets from NAT detection) on five days' datasets. In NAT detection experiments, AD Tree classifier stands out from the other two classifiers: SVM and J48, with an excellent performance on accuracy and practical detection on the real NAT device. Five attributes out of eight are suggested for effective NAT detection through experimental results. And an interesting finding about the NAT detection in the experiments is that the accuracy of NAT detection stayed high when the training dataset and the validation dataset were processed from datasets of similar sizes. Hence in order to accurately and effectively detect NAT devices on the network, a training dataset with the similar size to the target dataset should be selected.

For the host identification experiments, two evaluation methods were adopted. The host identification accuracy increased significantly as the change of four parameters. Among the four parameters, the improved connection separation method contributed the most to the host identification accuracy. At last, host identification experiments on datasets from the NAT detection stage are introduced. Host identification on the datasets from NAT detection stage detected few hosts. And the accuracy of these experiments was generally lower than that in the

experiments using the original datasets in Section 4.3. Except for a fewer number of packets in the dataset, this also might because that the deficiency of none HTTP packets in one TCP connection increases the chance of generating an inaccurate artificial line of TCP timestamp values and the system uptime values. Hence it is suggested to employ the NAT detection and the host identification separately as described in Section 4.2 and Section 4.3 on the target datasets to get a good performance.

Chapter 5 Conclusion

The main objectives of this thesis are: a) detecting NAT devices out of end hosts through supervised machine learning algorithms on HTTP attributes, and b) identifying end hosts behind the detected NAT device based on TCP timestamp values and system uptime of TCP packets. The research was performed on the datasets collected in our lab. Specially, this thesis provides an approach to combine the above two stages and improve their performances. Thus, this research can be used as a forensic analysis tool to analyze cybersecurity or other incidents that could occur on an organization's network from unknown NAT devices. Furthermore, the proposed framework only employs attributes from Transport layer and Application layer of a network, it can be employed on any Data Link layer and Network layer protocols.

The NAT detection employs the artificial NAT generation method proposed in Komˇarek's research to address the problem of lacking labelled datasets. Among the three different machine learning classifiers applied for NAT detection, the AD Tree classifier stands out with an excellent performance on accuracy and practical detection on the real NAT device. Time complexity and accuracy comparison of these three classifiers are shown in Table 5.1. This thesis also filters out the attributes with no influence on NAT detection. This left me with the most effective five attributes for NAT detection. The reduction of attributes improves the efficiency of preprocessing datasets for the machine learning classification. The NAT detection approach has been applied on five days' datasets collected in our lab, and a high performance was achieved on the experiments performed, where most of the accuracies observed were as high as 100%. In addition to the experiments where training and validation (test) datasets were captured on the same day, this NAT detection solution has also been employed in the experiments where the training dataset and the validation dataset were captured on different dates with different hosts running. This enabled me to test whether the proposed solution could generalize well over different hosts and traffic captured at different times / dates. Based on the difference of the accuracy, I concluded and verified an interesting finding that the NAT detection accuracy stays high when the training dataset and the validation dataset are processed from datasets of similar sizes. This discovery helps in selecting an appropriate training dataset according to the size of the target dataset in order to achieve an accurate classification.

Table 5.1 Comparison of three classifiers

ML Algorithm	Time Complexity	Best Accuracy on validation datasets	Real NAT detection
SVM	O(n^3)	92.86%	no
C4.5	O(n)	92.86%	no
AD Tree	O(n)	100%	yes

The host identification in this research is based on the fact that TCP timestamp values must be at least approximately proportional to real time. Host identification works in the way of generating artificial lines of TCP timestamp values and system uptime values by each connection and comparing the distance of the generated lines in a Python program. Connections are identified as belonging to the same host as long as the distance between two lines is smaller than a threshold value $\delta boot$. Two evaluation methods were adopted in host identification in terms of a) evaluating whether the connections that are identified to belong to the same host by the program are really sent from the same host, and b) evaluating whether original connections from the same host are identified to belong to the same hosts (might be several hosts) by the program. The host identification experiments were performed on the datasets processed from the NAT detection stage and the datasets processed directly from the original datasets. A series of experiments were performed in order to get the appropriate parameters for host identification, such as the limit for the number of packets in one connection, the distance calculation method between two lines, the connection separation method, and the different threshold values. The accuracy of experiments with those parameters improved from the accuracy of experiments with the settings in Georg's research on our datasets. The experiments on the datasets directly processed from the original datasets detected more hosts than the number of actual hosts in the datasets. The accuracy of these experiments using evaluation method A varied on different datasets from 34% to 89%. This reveals that a number of connections from the same host were detected as belonging to more than one host. While experiments on the datasets processed from the NAT detection stage detected fewer hosts than the number of actual hosts. The accuracy of them was lower than that of experiments on the corresponding datasets directly processed from the original datasets. Some of the hosts were not detected on those datasets. It is suggested to adopt the datasets processed from the original datasets for host identification, that is, performing the NAT detection process and the host identification process separately. The accuracy of experiments on both kinds of datasets using evaluation method B remained 100%, revealing that connections of any detected host were actually sent from the same end host. Under the two evaluation criteria, the host identification

approach identifies connections sent from the same host as sent from several hosts, but the connections of any identified host were indeed belonging to the same host. Since host identification behind a NAT device is aimed at identifying hosts with cybersecurity risks in order to manage them (such as blocking the malicious end hosts), the problem of identifying connections sent from the same host as sent from several hosts can result in blocking more than one detected host for only one actual host. This increases the workload but has no influence on accurately managing malicious hosts.

There are several lines of research arising from this research that could be pursued in the future:

I. The observation regarding the observation about the effect of dataset sizes on the NAT detection accuracy can be further verified by experiments on more datasets.
II. More experiments can be performed to find out the solution to improve the accuracy of host identification under evaluation method A.
III. More values between 100 ms to 600 ms can be tested to learn more about the influence of threshold values on host identification.

Bibliography

[1] M. Smith and R. Hunt, "Network security using NAT and NAPT," in *10th IEEE International Conference on Networks (ICON 2002). Towards Network Superiority (Cat. No.02EX588), Singapore, 2002, pp. 355-360*, Singapore.

[2] C. Engineers, "Network Address Translation (NAT) FAQ," 10 November 2014. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/network-address-translation-nat/26704-nat-faq-00.html#gen-nat. [Accessed 20 June 2018].

[3] T. Komárek, M. Grill and T. Pevný, "Passive NAT detection using HTTP access logs," in *2016 IEEE International Workshop on Information Forensics and Security (WIFS), Abu Dhabi, 2016, pp. 1-6*.

[4] E. Bursztein, "Time has something to tell us about network address translation," November 2007. [Online]. Available: https://cdn.elie.net/static/files/time-has-something-to-tell-us-about-network-address-translation/time-has-something-to-tell-us-about-network-address-translation-paper.pdf. [Accessed May 2017].

[5] G. Wicherski, F. Weingarten and U. Meyer, "IP agnostic real-time traffic filtering and host identification using TCP timestamps," in *38th Annual IEEE Conference on Local Computer Networks, Sydney, NSW, 2013, pp. 647-654*.

[6] V. Jacobson, R. Braden and D. Borman, "TCP Extensions for High Performance," May 1992. [Online]. Available: https://tools.ietf.org/html/rfc1323. [Accessed September 2017].

[7] N. Vratonjic, K. Huguenin, V. Bindschaedler and J. Hubaux, "A Location-Privacy Threat Stemming from the Use of Shared Public IP Addresses," in *IEEE Transactions on Mobile Computing, vol. 13, no. 11, pp. 2445-2457, Nov. 2014*.

[8] Y. Gokcen, V. A. Foroushani and A. N. Z. Heywood, "Can We Identify NAT Behavior by Analyzing Traffic Flows?," in *2014 IEEE Security and Privacy Workshops, San Jose, CA, 2014, pp. 132-139. doi: 10.1109/SPW.2014.28*.

[9] V. Krmicek, J. Vykopal and R. Krejci, "Netflow based system for NAT detection," in *5th International student workshop on Emerging networking experiments and technologies, 2009, pp. 23-24*.

[10] SolarWinds, "What is Netflow?," SolarWinds, [Online]. Available: https://www.solarwinds.com/what-is-netflow. [Accessed June 2017].

[11] S. AbtChristian, D. Baier and S. Petrović, "Passive Remote Source NAT Detection Using Behavior Statistics Derived from NetFlow," in *Emerging Management Mechanisms for the Future Internet, 2013, pp 148-159*.

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

[13] S. Mongkolluksamee, K. Fukuda and P. Pongpaibool, "Counting NATted hosts by observing TCP/IP field behaviors," in *2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, pp. 1265-1270*.

[14] A. Castiglione, A. De Santis, U. Fiore and F. Palmieri, "Device Tracking in Private Networks via NAPT Log Analysis," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Palermo, 2012, pp. 603-608*.

[15] N. Vincenzo Verde, G. Ateniese, E. Gabrielli, Luigi Vincenzo Mancini and Angelo Spognardi, "No NAT'd User left Behind: Fingerprinting Users behind NAT from NetFlow Records alone," in *2014 IEEE 34th International Conference on Distributed Computing Systems, 2014, pp. 218-227*.

[16] R. Weaver, "Visualizing and Modeling the Scanning Behavior of the Conficker Botnet in the Presence of User and Network Activity," in *IEEE Transactions on Information Forensics and Security, vol. 10, no. 5, pp. 1039-1051, May 2015*.

[17] Wikipedia, "iMacros," Wikipedia, 2017. [Online]. Available: https://en.wikipedia.org/wiki/IMacros. [Accessed May 2017].

[18] Wireshark, "Tshark," [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html. [Accessed June 2017].

[19] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in *IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998*.

[20] S. Ruggieri, "Efficient C4.5 [classification algorithm]," in *IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 2, pp. 438-444, March-April 2002*.

[21] L. Yuxun and X. Niuniu, "Improved ID3 algorithm," in *2010 3rd International Conference on Computer Science and Information Technology, Chengdu, 2010, pp. 465-468*.

[22] H. Sok, M. Chowdhury, M. Ooi, Y. Kuang and S. Demidenko, "Using the ADTree for feature reduction through knowledge discovery," in *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Minneapolis, MN, 2013, pp. 1040-1044*.

[23] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "Weka 3: Data Mining Software in Java," 2016. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/. [Accessed October 2017].