h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

- Fachbereich Informatik -

Passive Remote Detection of Network Address Translation (NAT) by using NetFlow

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Christian Dietz
February 21, 2013

Referent:	Prof. Dr. Harald Baier
Korreferent:	Prof. Dr. Slobodan Petrović
Betreuer:	Sebastian Abt, M.Sc.

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, February 21, 2013 _____

Christian Dietz

Acknowledgements

Foremost, I would like to acknowledge my supervisors Prof. Dr. Harald Baier, Prof. Dr. Slobodan Petrović and Sebastian Abt, M.Sc. for their great interest, help and guidance throughout my Master's Thesis. In addition, I especially would like to thank Sebastian Abt, M.Sc. for proposing the topic for this Master's Thesis and for acquiring the data sets from the Internet service provider, which were the foundation for many great results that I could achieve in this thesis. My deepest gratitude to all of you for being my mentors during the last 6 month.

Since this thesis mainly was written during my stay at Gjøvik University College (GUC), in Norway, I also especially would like to acknowledge the people who made this stay abroad possible. In particular I would like to thank Prof. Dr. Slobodan Petrović for being my second supervisor at GUC and for sharing his great knowledge during the last 6 month. Furthermore, I would like to thank the Center for Advanced Security Research Darmstadt (CASED) and especially Prof. Dr. Harald Baier, as a representative, for supporting my stay abroad financially by the CASED scholarship.

In addition, I would like to thank, the staff and fellow students at GUC faculty, who greatly enriched my academic experience and inspired me a lot during the time I spend in Gjøvik.

I also would like to thank rh-tec Business GmbH, for providing NetFlow data from two of their business customers.

Finally, I would like to thank my family for the enduring support throughout my studies and my whole live. Your love, advice and mental as well as financial support mean so much to me and have been priceless.

Darmstadt, February 2013
Christian Dietz

Abstract

The Internet has become one of the most critical infrastructures today. Our economy and society heavily rely on it. Because of the value of assets moved through the Internet, cyber criminals attack it every day and using botnets is a common attack method that they employ. To defend the infrastructure against botnet attacks it is important not only to detect them, but also to reliably estimate their strength. The strength of a botnet is mainly determined by the amount of computational power and network bandwidth, at which it has access to. Yet, as of today, botnet size estimation usually is done by counting IP addresses occurring in Command and Control (CnC) traffic. Recent research on real botnet command and control infrastructures showed that estimating a botnets size based on IP addresses is rather imprecise. This is due to the fact, that multiple infected devices often share one internet line and IP address. Therefore, detecting NAT is the first step towards a more precise estimation of the size of a botnet.

Nevertheless, network security threats do not only exist on Internet scale. They for example also exist in local enterprise networks. In these networks, employees might setup own unauthorized routing devices, for example to share their single official internet connection among their workstation and their mobile devices. Setting up rogue access points in these networks is a major security threat, because these devices are out of the scope of any security policies and might provide an easy to exploit attack vector. Since such a devices normally also shares one IP address among different devices, it has to implement some sort of NAT-Gateway. Therefore, by being able to detect NAT, these rogue access points can be detected as well.

This thesis demonstrates new efficient and reliable methods for NAT detection based on NetFlow data. Previous NAT detection approaches have in common, that they usually rely on special data fields of a packets IP or TCP header, therefore relying on deep packet inspection. In contrast to this, the methods, which are presented in this thesis, only depend on highly aggregated communication statistics derived from NetFlow data. Therefore, these detection methods are very efficient, can be performed in any network deploying state-of-the-art network equipment and at very high traffic rates. In this thesis, three different approaches have been developed and tested on simulated data as well as on NetFlow data retrieved from two business customers of a German ISP. These approaches are based on passive OS finger printing, the detection of incremental usage of source ports and on distinguishing single user from multiuser behaviour based on machine learning techniques. Finally, in this thesis also proposes a Decision

Fusion Concept for combining these distinct classifiers into one system. This system will be able to efficiently and reliably detect NAT on normal as well as on abnormal (i.e. malicious) traffic.

Kurzfassung

Das Internet ist zu einer der wichtigsten Infrastrukturen geworden. Unsere Wirtschaft und Gesellschaft befinden sich in einem starken Abhängigkeitsverhältnis zu dieser Ressource. Aufgrund der Höhe von Vermögenswerten, die durch das Internet bewegt werden, wird es täglich von Cyber-Kriminellen attackiert und Botnetze sind ein häufig verwendete Angriffsmethode für diese Attacken. Um diese Infrastruktur gegen Botnetzangriffe zu verteidigen, ist es wichtig diese Angriffe nicht nur zu erkennen, sondern auch zuverlässig ihre potenzielle Angriffsstärke abzuschätzen. Die Stärke eines Botnetzes wird im Wesentlichen durch die ihm verfügbare Rechenleistung und die Netzwerkbandbreite, auf die es Zugriff hat, bestimmt. Allerdings wird diese Abschätzung heute in der Regel auf Basis eines einfachen Abzählens der IP-Adressen vorgenommen, welche aus dem sogenannten Command and Control (CnC) Netzwerkverkehr ermittelt werden. Aktuelle Forschungsergebnisse zeigten, dass die Abschätzung der Größe eines Botnetzes, welche auf dem Abzählen der IP-Adressen beruht, ziemlich ungenau ist. Der Grund dafür ist, dass mehrere infizierte Endgeräte häufig dieselbe Internetverbindung und IP-Adresse nutzen. Deshalb, ist die Detektion von NATs der erste Schritt hin zu einer präziseren Abschätzung der Größe von Botnetzen.

Jedoch existieren Netzwerksicherheitsrisiken nicht nur im Internet, sondern beispielsweise auch in Unternehmensnetzen. In diesen Netzwerken können Angestellte unerlaubterweise ihre eigenen Router installieren, um zum Beispiel ihren offiziellen Internetanschluss ihres Arbeitsplatz PCs auch mit den privaten mobilen Endgeräten zu teilen. Das Installieren solcher unautorisierten Router stellt ein erhebliches Sicherheitsrisiko dar, da sie sich außerhalb jeglicher Sicherheitsrichtlinie befinden und nicht selten einen einfach auszunutzenden Angriffsweg für einen Angreifer darstellen. Da diese Geräte üblicherweise eine IP-Adresse verschiedenen anderen Geräten gleichzeitig zur Verfügung stellen, müssen sie ebenfalls die Funktionalität eines NAT-Gateways bereitstellen. Daher ermöglicht es die Detektion von NATs, auch diese unautorisierten Zugangspunkte zu erkennen.

Diese Abschlussarbeit demonstriert neue effiziente und zuverlässige Methoden für die Detektion von NATs basierend auf NetFlow-Daten. Bekannte NAT Detektionsansätze haben gemeinsam, dass sie normalerweise auf speziellen Datenfeldern von IP oder TCP Headern einzelner Pakete basieren und daher auf der sogenannten "Deep Packet Inspection" basieren. Im Gegensatz dazu basieren die in dieser Thesis vorgestellten drei Methoden ausschließlich aus NetFlow Daten gewonnenen hochgradig aggregierten Kommunikationsstatistiken. Darum

sind diese Methoden sehr effizient und können in jedem Netzwerk verwendet werden, in dem aktuelle Hardware zum Einsatz kommt, auch bei solchen mit hohen Datentransferraten. Im Rahmen dieser Abschlussarbeit wurden drei unterschiedliche Methoden zur Detektion entwickelt und auf simulierten Daten, sowie auch auf NetFlow Daten von zwei Geschäftskunden eines deutschen Internet Providers, getestet. Darüber hinaus wird in dieser Abschlussarbeit auch ein Fusionskonzept vorgestellt, mit dem diese unterschiedlichen Klassifikationsmethoden in einem System vereint werden können. Dieses System wird in der Lage sein NATs effizient und zuverlässig, sowohl auf normalem als auch abnormalem (z.B. bösartigem) Netzwerkverkehr, zu erkennen.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter provides an overview of the motivation, related work, the objectives and the structure of the thesis.

1.1 Motivation

Network address translation (or short: NAT, RFC1631) offers the possibility to hide a network infrastructure behind one single IP address or a limited pool of IP addresses. Using this technique has become very popular with the growing of the internet and the resulting shortage in IP v4 addresses for subscribers. By using NAT it is possible to hide a complete local network behind one single IP address like it is just one "normal" device (from the service provider's perspective), which reduces the number of addresses that have to be provided on internet scale.

Nevertheless this technique does not just provide benefits. The possibility to hide a network can also be used to set up unauthorized sub networks, like for example an unauthorized Wi-Fi access point in an enterprise network. Some References such as [[13]] say that this is a growing concern, since the number of handheld devices in enterprises is increasing. The reason for this concern is that most of these devices normally provide 3-/4G and Wi-Fi to access the internet. To reduce the fees, that normally have to be paid for 3-/4G internet access, it could be very tempting for employees to set up their own Wi-Fi access point to connect their device to the internet.

This access point then uses NAT to connect multiple devices to the enterprise network as if they were just one device (from the administrator's point of view). The problem with that is that these devices are usually unauthorized and therefore are out of the scope of any professional it-security policy. Taking this into account, it is obvious that the ability to detect NAT could be very important to detect unauthorized NAT devices such as Wi-Fi access points in a corporate network. It is obvious that these devices provide potential attack vectors for industrial espionage or any other cyber-attack against the corporate network. Regarding [[13]]

one has also to consider that these handheld devices could be both connected via 3-/4G and Wi-Fi in parallel. That means that these devices could potentially bridge their native broadband connection into the Wi-Fi connection that has been introduced onto the corporate network" [[13]].

In the previous paragraph the motivation for NAT detection was more described like a technique to prevent cyber-attacks by detecting unauthorized usage of NAT in corporate networks before any attack has happened. But besides this, NAT detection could of course also be very useful in network forensic analysis, e.g. to investigate, if there was an unauthorized NAT device. This could even be done if the device is not present any more, by using recorded network traffic.

Furthermore, detecting and counting NAT can also be helpful when it comes to estimate the size of Botnets. A Botnet is an unauthorized network that consists of malware infected hosts that are controlled by an unauthorized control instance such as a Command and Control Server (C&C-Server). Botnets are often used for criminal activities such as distributing Spam E-Mails or performing Distributed Denial of Service Attacks (DDoS Attack).

As B. Stone-Gross et al. described in [[42]], without being able to detect a NATed sub network behind an IP address the real size of a botnet is likely to be underestimated:

> "We observed 9,336 distinct bots for 2,753 IP addresses from these infected machines on private networks. Therefore, if the IP address count was used to determine the number of hosts it would underestimate the infection count by a factor of more than 3 times."

That means, estimating the size of a detected botnet by just counting the IP addresses which are subscribing to a particular C&C-Server is not the same as counting the actual infected hosts. This becomes obvious when keeping in mind that most residential networks consist of more than one device (host) with access to the internet. These devices normally subscribe to the internet via some sort of NAT-Gateway, for example a wireless router. But of course each of these hosts in the residential network could be infected with a piece of malware that makes it a Bot, which is then a part of a certain Botnet. Bearing this in mind, it is obvious that the possibility to detect a NATed sub-network behind a certain IP address and estimating the size of such a sub-network contributes to a much more precise size estimation of a Botnet.

Chapter 2 will give an overview of the state of the research that has already been done on the topic.

1.2 Objectives

In the previous section, the motivations for detecting NAT as well as two main use cases for the NAT detection were presented. This section outlines the concrete objectives of this thesis.

One of the main objectives of this Thesis is to find features relevant for detecting NAT only based on NetFlow. That means that the detection should only be based on a very limited data set such as the Source-IP, Destination-IP, Source-Port, Destination-Port, Layer 4 Protocol plus the timestamp, in the optimal case. In addition to these data fields, the standard NetFlow datagrams contain further data fields, for example MAC addresses or the number of incoming and outgoing packets and the amount of incoming and outgoing traffic. The whole data set is described in Chapter 2. This also includes a discussion for each field whether it is theoretically usable for detecting NAT and/or estimating the size of a network behind a NAT or not.

The benefit of working on this dataset is in the fact that there is no confidential user payload involved, storage costs are reduced to the absolute minimum and there are almost no extra costs for extracting the data, due to the fact that today almost all routers can provide NetFlow data (at least in the ISP domain).

Another important objective of this thesis is to analyze, whether it is possible to detect NAT only based on NetFlow data, or not. Based on the developed features concrete classification algorithms were developed and analyzed regarding their usefulness. These algorithms could for example be based on neural networks, support vector machines or any other machine learning method.

When it comes to the selection of possible features in this context, then it is also important to keep in mind that there are at least two possible ways in which one of the data fields in the data set could become a feature for detecting NAT. The first possibility is that the field has to be changed during the process of doing the network address translation by the routing device (e.g. IP addresses). The other possibility is that a data field is influenced by the user behavior (e.g. time patterns in the flows, payload size, etc.). Bearing this in mind, the objective of selecting features is divided into two distinct objectives. One objective is to find feature in the data fields that are changed due to the process and the other objective is to find features by considering the user behavior.

In order to find features by considering the user behavior, a model has to be constructed that can distinguish between multiple NATed hosts (users) and one single non NATed host (user). In contrast, a second description could be necessary to describe multiuser behavior. When multiuser behavior is detected on a single IP then it can be considered a NAT.

This analysis also involves the generation of suitable test data. In order to be able to produce different user profiles (single or multiuser) it is necessary to have a fully controllable and flexible environment to extract data from. Therefore a further objective of this thesis is to set up a simulation environment and extract different NetFlow samples. The benefit of this approach is that NetFlow data can be captured before and after the NAT gateway, which makes it easier to find differences (features). After generating test sets a statistical analysis could be helpful to recognize further features.

The next objective is to test the classifier(s) on real NetFlow data from an internet service provider (ISP). To be able to decide, if the classification is also correct on real ISP data, the ARP-Tables of the Routers are also extracted to have a rough overview of the Network topology.

Another goal of this thesis was contribute to a more precise estimation of the size of a network, which is hidden behind a NAT gateway. This objective, heavily depends on the results of the first step, because obviously without being able to detect NAT, it would be impossible to estimate the size. For both, the detection and the size estimation, it is necessary to analyze the quality of the classifiers (e.g. by determining their detection rates). The Arp-Tables can support the quality analysis, because they contain the MAC-Addresses of the network tokens behind the NAT gateway. Hence, the Arp-Tables will allow just a rough approximation of the size of the Network, because there could be a complex infrastructure with cascaded NATs.

Summary of objectives of this thesis:

1. Extracting new features.
2. Building classifiers on different levels of the communication.
3. Developing a Simulation environment for retrieving sample data.

4. Analyze, if approaches applied on other data sets are also applicable on NetFlow.
5. Analyze, if the NAT-Gateway itself introduces patterns to the communication.
6. Showing NAT detection in a real world scenario.

1.3 Structure of the Thesis

Now that the motivation and objectives of this thesis were presented, the remaining parts of this thesis are structured in the following way.

Chapter 2 describes the necessary fundamentals of network address translation (NAT) and NetFlow that are some of the most important concepts on which this thesis is based on.

In Chapter 3 the fundamentals of feature selection, classification and the requirements on a Simulation environment are described in detail. This chapter describes the necessary background knowledge of the field of machine learning and lines out, which fundamental requirements a simulation in general and particularly in this context has to fulfill.

Chapter 4 then specifies how the data that is used later in this thesis was generated. This includes a description and discussion of the chosen simulation environment as well as a description of the capturing and sampling of the ISP dataset.

The following Chapter 5 then provides an analysis of these data sets and concludes with an outline of possible features representing the results of this analysis. These results also include a description of possible necessary preprocessing steps.

In chapter 6 these features are used to build classifiers and the preprocessing as well as the filtering are applied and tested in practice. The results are presented and discussed at the end of the chapter.

The final chapter 7 provides a short summary, outlines the contributions of this thesis and describes open points for future work in this field.

CHAPTER 2

Foundations of NAT-Detection and NetFlow

This chapter provides an overview of related work that has already been done by others as well as foundations of the underlying technologies of this thesis.

After the related work section, the next topic covered by this chapter is the general process of network address translation (NAT) as well as its different setups. This then is followed by the definition of a Network Flow is, and how these Flows are sampled extracted and captured in practice. The next important topic covered by this chapter is the Feature selection theory, which is essential in every system that tries to detect certain patterns. Then, knowing how the feature selection works in theory, the next step in theory as well as in practice is to use the selected feature (set) to build classifiers that then could be used to automatically classify/detect NATed Network traffic. These classification approaches could be based on expert knowledge, on machine learning techniques or a combination of both. The last part of this chapter will then be a theoretic description of the requirements of a Simulation Setup that was used for retrieving sample data and training of the classifiers.

2.1 Related work

NAT detection has already been treated in the literature.

For example [[47]] provides an overview of different techniques used to detect NAT in the field of network forensics. Regarding this resource, one has to distinguish between active and passive detection of NAT. The active methods described are "port scanning" and "routing test". The port scanning method is based on special management ports that are used by certain software NAT Firewalls. The routing test method is only capable to detect NAT in case of a misconfigured network address translator, by contacting an external web-resource with modified routing tables.

According to [[47]], passive techniques that are known are "leaked real IP address", "strict source port translation", "OS fingerprinting" and "IP TTL". The first technique is based on the leakage of the real IP address in packets of some higher level protocols such as for example SMTP or DNS, while network address translators only change the IP headers. The OS fingerprinting method is based on the fact that a single host will not produce fingerprints of several hosts at the same time. This method could be extended by using IP IDs and TCP timestamps. The IP TTL method is based on general behavior of network address translator to reduce the IP TTL values of transmitted packets. The IP TTL method is described in [[27]] in more detail. It is important to mention that, in contrast to other systems, that perform a deep packet inspection, this system operates only on sFlow (RFC 3176), which means that the system operates on very limited data.

In [[19]] a system architecture and three new methods for NAT detection are described. The system described there mainly operates on NetFlow, but with three additional data fields in each record. These fields are "TTL", "IP ID" and "TCP SYN packet size". The values of all three parameters are extracted from the first packet of a new flow. This paper gives some useful inspiration on how NAT detection could be done, but it is important to mention that the described system seems to be rarely tested and only works on a very small simulated environment.

According to Kohno [[18]], clock skews of different physical hosts can also be used as an indicator for detecting the presence of different physical hosts at the same time. Their approach is based on the observation, that the hardware clock of a host has measurable skews, which also influence the timestamps in the device's TCP packets, which are then results measurable. It is important to notice that this approach, as well as many others, is based on specific header fields of network packets. This is in contrast to the goal of this thesis, which is to detect the presence of a NAT-Gateway by passively analyzing NetFlow data. This NetFlow data is a very high-level aggregated description of the communication between specific hosts.

As described in the motivation chapter, detecting a NAT can be very useful, but when it comes to a forensic investigation, where the scale of a detected unauthorized network is of interest, just detecting it is not enough. One method to estimate the size of a Network behind NAT is described by S. Bellovin in [[3]]. The paper describes an algorithm that is only based on the IP ID field and furthermore gives an overview of some problems one might face, when using this method with real life data.

Moreover L. Rui et al in [[32]] and [[20]] presented a detection approach that is not based on any special field of a packets header. Their method is based on 5 Assumptions, which results in 8 features, which are based on counting the number of packets with certain traits, such as for example the number of DNS-Requests, or the overall amount of packets sent out during a certain time window. This approach is particularly of interest for this thesis, because most of their features can also be extracted from NetFlow data. Some of them are even directly extractable, like for example the number of outgoing packets during a certain timespan. Hence others, like the number of Fin- and SYN-packets are not directly applicable, because since NetFlow data is an aggregation of the communication behaviour. These packets will most likely appear among others and therefore they will often be aggregated into one flow record. Nevertheless, there is still a chance to extract this information, since at least their presence will be indicated in the corresponding flow records.

After this overview of work related to the detection of NAT, the next chapter provides a detailed description of the network address translation concepts.

2.2 Network Address Translation

Network address translation (NAT), as described in RFC2663, in general is a technique for modifying address information in IP headers of network packets. Basically, each NAT-device interconnects two parts of a network. These are usually called "inside" and "outside". The term "inside" typically describes interfaces of the Gateway on the private local part of a network. This part consists of a number of hosts, which are connected to the NAT-Gateway from the private network, of which the IP addresses are modified or translated respectively. The term "outside" refers to the interface(s) of the NAT-Gateway that provide the connection to another network (private or public) with the translated IP address(es). In most configurations, the outside part usually is the World Wide Web.

This principle is also illustrated in figure [2.1]:

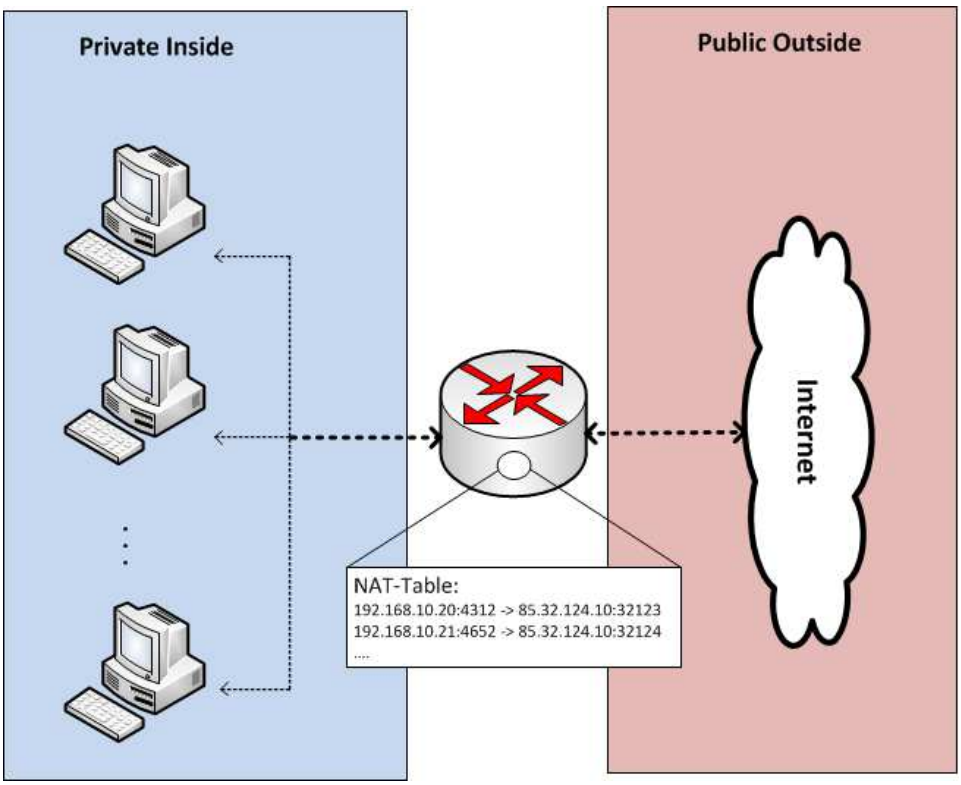


Figure 2.1: Basic Concept of NAT.

In the most basic configuration a NAT device could map each "inside" IP address to a corresponding "outside" IP address. In more complex configurations, NAT is normally used to map a whole subnet on the inside to a single IP address or a limited pool of IP addresses on the outside. This technique helped to solve the problem with the shortage in IP v4 addresses for a while. Furthermore, it is often used in private consumer networks. The reason for this is, that private customer's usually only get one single IP address assigned by their Internet service provider at a time. Thus most customers today have more than one device with Internet connectivity. To be able to connect all of these devices at the same time via the same IP address to the World Wide Web, it is necessary to place a NAT-Gateway in-between these devices. Thus, the residential Internet access point has to share this one public IP amongst all these devices. Today, the Internet access point and the NAT-Gateway are usually integrated into one single device - especially in network equipment of private consumers.

2.2.1 Preliminaries

A NAT device/gateway has to do several things to make the sharing of one single IP address possible. For example, it has to change several data fields in each IP packet traversing from the inside to the outside or vice versa.

One obvious data field that has to be changed is the Destination IP address (for short: Dst-IP) of incoming packets and the Source IP address (for short: Src-IP) of outgoing network packets. Furthermore the IP ID field is changed and the time to live of the packet is decremented by one, which is always done when the packet passes one hop.

Since each active host from inside the network will probably use several ports it is likely to happen that more than one host tries to use a specific port at a time. Because of this, the NAT Gateway has to be able to change these port numbers to another (free) value. This process is called Port Address Translation (PAT).

Hence, when changing particular fields of an IP packet, it is also necessary to re-compute the checksums of the packets after these changes. This is necessary, because otherwise the hash value computed by the receiver wouldn't fit to the one contained in the packet, which would indicate that the packet content was changed/corrupted (e.g. through transmission errors). In such a case the receiver then would either drop the packet or ask for a retransmission, depending on which protocol was used.

Even knowing that Network Address Translation and Port Address Translation are technically two different things, it is common to just talk about NAT, but meaning the combination of NAT and PAT. When talking about NAT, it is also important to know the commonly used taxonomies in this field. Different interest groups have developed their own taxonomies for describing different NAT setups according to their intent, on of working with them.

For example, in the documentation of huge network equipment manufacturers such as CISCO, the before mentioned combination of NAT and PAT is also often referred to as NAT overload [[48]] as one of three fundamentally different ways to setup a NAT. This source of information represents the administrator's point of view. Further, well known NAT setups in this domain are static NAT and dynamic NAT.

The "Session Traversal Utilities for NAT" (STUN, RFC5389 and RFC3489) is another group of interest that developed its own taxonomy for classifying different types of PAT setups and providing methods for traversal of each of these types. The STUN group/protocol aims to solve several problems that are introduced by NAT Gateways in several applications such as peer-to-peer or VoIP applications by implementing NAT traversal methods.

In the following sections, the before mentioned NAT topologies are described in more detail.

2.2.2 Static NAT

Static NAT is the most basic configuration of a NAT. It simply maps one inside IP address to one outside IP address. Hence hosts on the inside are reachable from the outside on designated ports.

This method is useful for example to connect Servers to the Internet that provide a certain service, which should be reachable via a certain IP address.

In case such a Server goes down (e.g. because of a hardware fault) the NAT Gateway could immediately change the mapping of the outside IP address to a backup server on the inside interface. In general, the main reason for using this configuration is to hide the real IP address but still keeping a network resource (e.g. a Server) accessible via a fixed IP address, which is necessary for some applications.

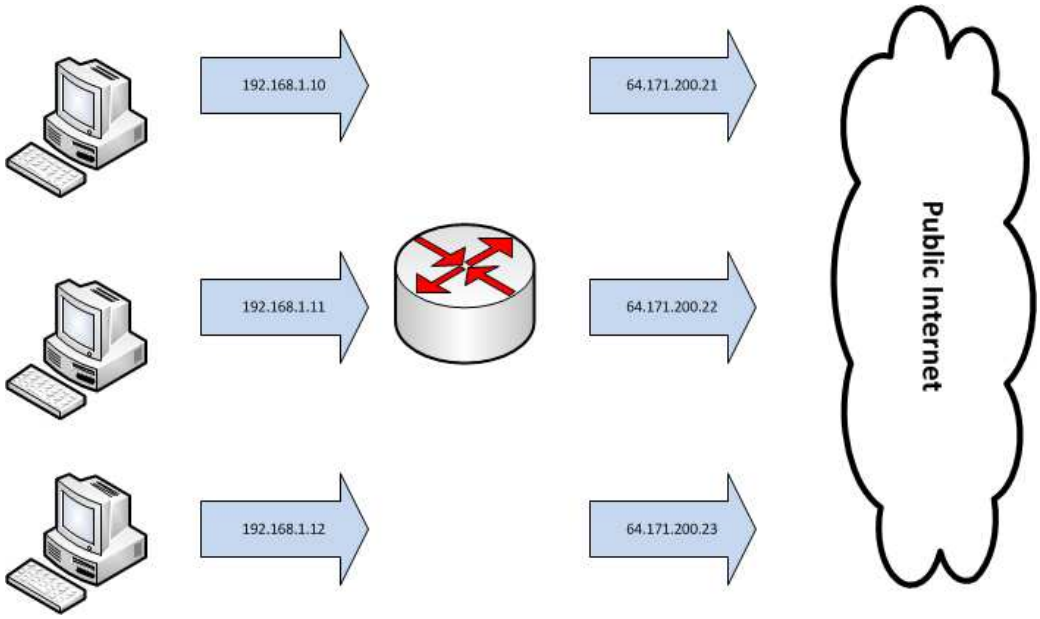


Figure 2.2: Principle of static NAT

As shown in Figure 2.2, in this setup only the IP addresses are changed/translated by the NAT-Gateway while the port numbers stay untouched.

2.2.3 Dynamic NAT

Dynamic NAT is an address translation method that translates an IP address from the inside to one IP address from a pool of public IP addresses.

Technically, the dynamic NAT-Gateway organizes the pool of public IP addresses in a table and the currently used private IP addresses on the inside in another Table. In case an IP address from the inside tries to setup a connection to the outside, the NAT-Gateway dynamically chooses a currently unused IP address from the table of the outside IP address pool and sets up a mapping between both addresses. This behaviour is also called a "many-to-many mapping". This principle is illustrated in Figure 2.3

Furthermore, this approach is also often used as a technique to enhance security of a network, because the mapping changes from time to time and therefore it is more difficult to attack one particular host on the private inside network. Moreover this approach makes it harder to passively monitor the individual usage patterns because just monitoring one IP address from the outside is not the same as monitoring one host. Even though applying this technique has the advantage that it can be seen as some sort of a security enhancement, it also has some serious drawbacks. The first drawback of this approach is that it does not hide any open application ports, which could be a potential security risk if they are not blocked in a different way. Additionally, it is obvious that this technique is only applicable if the resource

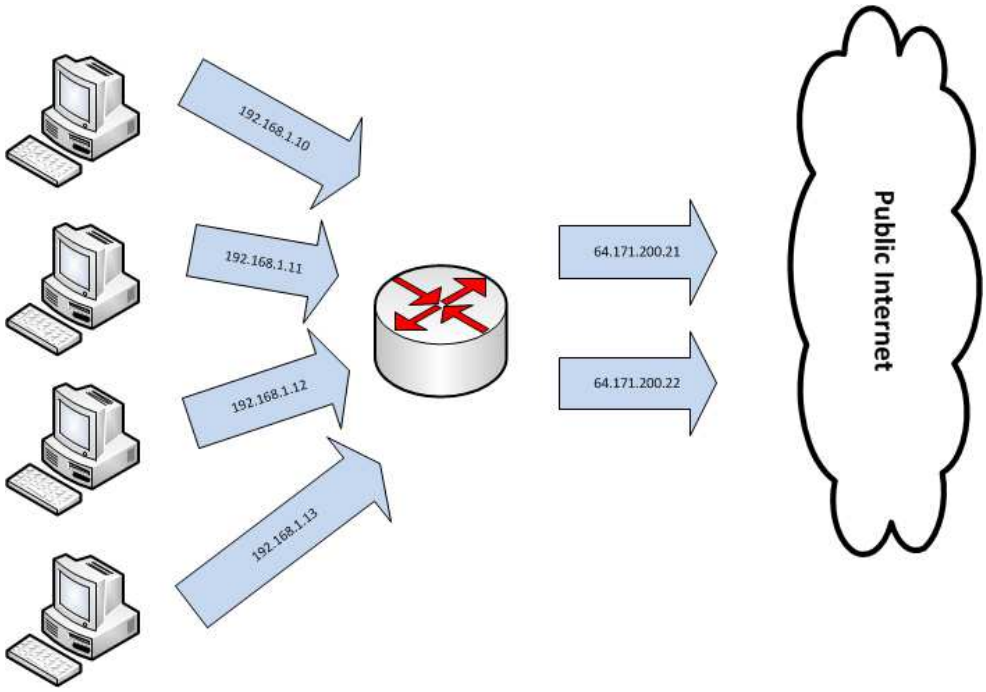


Figure 2.3: Principle of dynamic NAT.

on the inside is not a service that has to be reachable via a fixed IP address from the outside.

2.2.4 NAT Overload / PAT

This is probably the most often used configuration of a NAT-Gateway - at least in private residential usage. In general, port address translation (aka: PAT) is a technique that translates port numbers, which are used to address the network sockets of certain applications after an IP packet has reached its destination host by using the IP address.

In general PAT can - the same way as the address translation - also be performed in a static and a dynamic way. Hence the dynamic translation is likely to be more often used in practice, because routers for private residential networks normally implement the dynamic port address translation on one public IP address on the outside interface. According to RFC6056, the dynamically assigned ports should be chosen from the Range between 1024 and 65535. The ports below are reserved for special application protocols. Figure 2.4 illustrates this principle.

In Figure 2.4 it can also be seen that the Router / NAT-Gateway has to keep track of the active translations in a special translation list.

As mentioned before, PAT is normally used along with dynamic translation of multiple inside IP addresses to one single public IP address on the outside. This technique is also known under the name NAT Overload, IP masquerading and many-to-one NAT. Hence following the suggestion of RFC 2663, this type of NAT the correct term for this configuration is NAPT (network address and port translation).

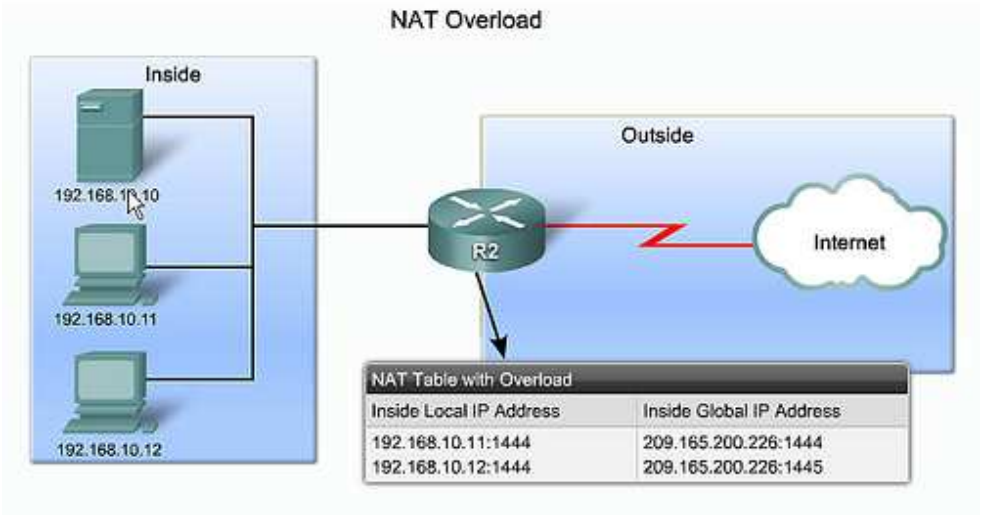


Figure 2.4: Principle of NAT Overload (NAPT).

2.2.5 Difference Source-NAT and Destination-NAT

In practice, network address translation is mostly implemented as the so called Source-NAT. All the before mentioned configurations belong to this class, because Source-NAT in general describes a way of network address translation, which maps inside ports dynamically to a (random) port on the outside interface(s). Hence, Destination-NAT does not follow this pattern, because Destination-NAT forwards packets received on a specific fixed port on the outside interface of the gateway to specific IP address connected to its inside interface.

In general, this concept is relevant for the further parts of this thesis, because it could be applied on the same NAT-Gateway in parallel to dynamic NAT. Therefore, it could have some influence on the patterns seen regarding the port numbers. This effect is of course no problem in the simulation because it can be controlled there and switched on or off respectively, but it could eventually be a problem when working on the ISP data set because these details about the underlying configuration the sampled data set is retrieved from will not be known. Even though it is normally not enabled in most standard configurations, almost every Router on the private consumer market today supports this feature. For example, it is sometimes necessary to enable it because some online games require receiving their incoming data packets via a certain port.

In general, it is important to know that this feature exists and that it might have some effect, but it is not considered to be a significant problem, because it will not be present in the simulation. Even if it will be present in the ISP data set, its emergence is assumed to be rather rare compared to the remaining traffic.

2.2.6 Problems caused by NAT

In the previous sections, the fundamental principles of NAT and the different configurations were described as well as the advantages each of these configurations potentially has. Hence NAT not only solves the problem with the shortage of IPv4 addresses on the Internet, but also introduces new problems under certain circumstances. For example, the presence of a dynamic NA(P)T causes some serious drawbacks for point-to-point applications such as many Voice-over-IP applications. This is obvious because these configurations masquerade the actual

physical hosts behind dynamically assigned IP addresses and/or ports. Therefore it is much more complicated to setup these communication channels directly between two hosts.

Because of these problems, several methods were developed to subvert the NAT-Gateway. One commonly used method is the so called STUN (Session Traversal Utilities for NAT) protocol, which is described in RFC5389 and RFC3489.

STUN (Session Traversal Utilities for NAT) is a protocol for detecting the presence of firewalls and NAT-Gateways and subverting a detected NAT-Gateway. The current version of the protocol is described in RFC5389. Even though it is possible to detect NAT by using this protocol, it does this on the host itself thus it detects NAT on the inside interface of the NAT-Gateway, in contrast to the approach presented in this thesis, where remotely detecting a NAT-Gateway on its outside interface is the main goal.

2.3 Network flows

NetFlow is a standard for high level monitoring and accounting of network usage introduced by CISCO. Several versions were introduced, such as for example v5 (RFC3954) or v9 (RFC3954). An overview of the Cisco NetFlow standards can be found in [[8]]. In addition, other Flow standards from other manufacturers or interest groups were introduced over the time, such as for example sFlow[[37]].

The main goal of this thesis is to show a novel approach for NAT detection from an ISP or enterprise network administrator's point of view solely based on NetFlow data. NetFlow was chosen as the relevant standard because, as mentioned before most ISP networks, as well as most enterprise networks, are based on CISCO products, which implement this standard. Detecting subnets for the sake of a more precise botnet-size-estimation as well as detecting unauthorized subnets are the two main use cases addressed by this thesis.

This section defines the term flow and provides a general overview of how these flows are generated and in which formats they are exported.

2.3.1 Preliminaries

Independent of the actual flow standard that is used, they all have in common that they provide a statistical overview over the network and therefore help to monitor the "health" of a network. The essential advantage of these techniques is that they all provide an additional "abstraction layer". This simply means that an administrator does not need to inspect particular packets to get information about the current state of a network. Hence, there are huge differences between the definitions what a flow is in the different standards.

While some of the flow-standards only provide aggregated (statistical) information about the communication parties at a time, others also capture and export parts of the transmitted content. The export of user content from packets provides valuable information for anomaly or misuse detection in general, but at the same time this can be a problem when it comes to data privacy regulations.

Since the goal of this thesis is to meet these data privacy regulations and not to rely on any user content, only standards that provide a very high level of abstraction are taken into account in this thesis. Besides this constraint, the proposed solution of this thesis should require as little extra equipment as possible and rely on standards that are already commonly

used in practice in networks of Internet providers.

Using flows as data sources for detecting NAT is favorable because the infrastructure for sampling, exporting and collecting this data is already present in most ISPs as well as enterprise networks. Therefore only the detection module itself has to be added. This is obviously very cost efficient and therefore there will probably be a low inhibition threshold to implement this detection approach in practice. This is an important argument in particular for the use case of botnet size estimation on the Internet providers side mentioned before.

2.3.2 Definition

The most basic definition of a flow is a 6-tuple consisting of the source IP address, the destination IP address, the source port, the destination port, the protocol used, and a time stamp. This is basically the most essential data to establish and describe a communication between two parties. This information is present independent on which flow standard is chosen. Hence, as described before, the intention of sampling and exporting flows is to provide an aggregated statistical overview of the current network usage. Therefore, each of the Flow standards extends this 6-tuple by more or less additional information.

As described before, the standard chosen for this thesis was Cisco NetFlow (v5 and v9), because it perfectly meets the before mentioned requirements of being commonly used in provider networks and meeting the data privacy regulations.

For example, the Cisco NetFlow standard in version 9 consists of 42 data fields, which most of them could (but need not) be set or left blank, depending on the actual use case and network environment applied to. For this thesis, the statistics about incoming and outgoing packets and the corresponding number of incoming and outgoing bytes as well as information about the TCP flags that were present during the sampling time window will be the main source of information. Further information that might be helpful are the more fine-grain timestamps in the Cisco NetFlow standard, because they provide information about the start time of a flow, the duration as well as a separate time stamp field for the end of the flow.

In total this leads to the 13-tuple (time start[sec], time end[sec], duration [msec], Src. IP, Dst. IP, Src. Port, Dst. Port, Protocol, TCP-Flags, num in packets, num in bytes, number out packets, number out bytes) as the main data source used in this thesis and extracted from the NetFlow standard.

The following figure 2.5 provides an overview of the principal working scheme of Netflow. In addition, it illustrates some of the main differences in the way flows of each of these two NetFlow versions are exported.

As it can be seen in figure 2.5, there is at least one export source of the flows and one collector destination. In this domain, the term "FlowSet" is commonly used to describe a set of flows that is transferred within one datagram. Figure 2.5 also points out that the main difference between the two versions is that version 5 is of fixed format. In contrast to this, NetFlow in version 9 is designed to be very flexible and therefore it can be extended by additional information depending on the intended use case. NetFlow version 9 uses 4 different types of records. First of all, it sends a flow template record. This template is used be to define the exported FlowSet data fields, which is the second type of records defined by this standard. In addition to this, NetFlow v9 implements a so-called option template and a corresponding options data record. These option FlowSets contain meta-data, to describe the flow exportation process and do not contain any flow information themselves. Since the detection approach presented within this thesis is based only on the actual flows, these options of NetFlow version

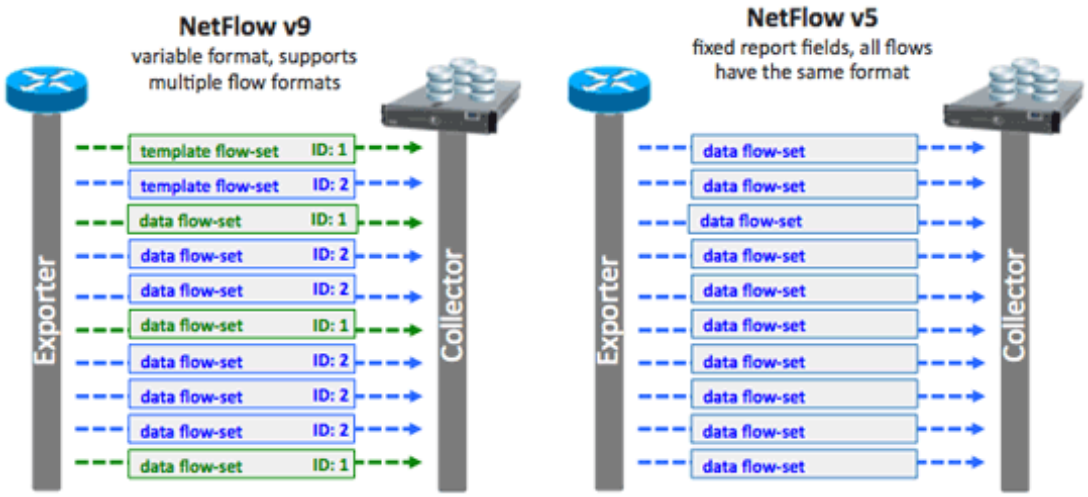


Figure 2.5: Overview differences and exportation of NetFlow v5 and v9. [[40]]

9 will not be considered in the further thesis.

Table 2.1 provides an overview of the NetFlow version 5 FlowSet data format:

As it can be seen from the table 2.1 the NetFlow version 5 Flow Record consists of 48 Bytes in total. The table also provides a detailed overview of the exported Data fields by this standard. Hence, it is also important to keep in mind that this record will be transferred as payload of a NetFlow datagram for transferring it from the flow exporter to the flow collector.

In contrast to the detailed description of the fixed NetFlow version 5 Data FlowSet given above, the description for the version 9 Data FlowSet is provided in a much more general way, since the format is not fixed. The figure 2.6 illustrates the scheme of the Data FlowSet for NetFlow version 9.

It is obvious that this flow format is designed to be very flexible and therefore the actual attributes that are contained in this format might differ depending on the actual use case.

Furthermore, it is important to point out that NetFlow version 5 is only applicable on IPv4 network traffic and is designed to provide unidirectional flows. This is in contrast to NetFlow v9, which is capable of sampling bi-directional flows and is also applicable on IPv6 networks.

More details about the exportation and collecting process follow in the last section of this chapter. The next section describes the details of sampling packets to flows. Details on how these flows could be transformed into features and be used for detecting NAT are presented in the following chapters.

2.3.3 Flow Sampling

The process of carrying out statistics about transferred packets between each host connected to the inside interface of the NAT-Gateway and the Remote Resource on the outside interfaces is called Flow sampling. As described before, for each connection in Internet addressing there

Bytes	Contents	Description
0-3	srcaddr	Source IP address
4-7	dstaddr	Destination IP address
8-11	nexthop	IP address of next hop router
12-13	input	SNMP index of input interface
14-15	output	SNMP index of output interface
16-19	dPkts	Packets in the flow
20-23	dOctets	Total number of Layer 3 bytes in the packets of the flow
24-27	First	SysUptime at start of flow
28-31	Last	SysUptime at the time the last packet of the flow was received
32-33	srcport	TCP/UDP source port number or equivalent
34-35	dstport	TCP/UDP destination port number or equivalent
36	pad1	Unused (zero) bytes
37	tcp_flags	Cumulative OR of TCP flags
38	prot	IP protocol type (for example, TCP = 6; UDP = 17)
39	tos	IP type of service (ToS)
40-41	src_as	Autonomous system number of the source (origin or peer)
42-43	dst_as	Autonomous system number of the destination (origin or peer)
44	src_mask	Source address prefix mask bits
45	dst_mask	Destination address prefix mask bits
46-47	pad2	Unused (zero) bytes

Table 2.1: Overview NetFlow record format. [[6]]

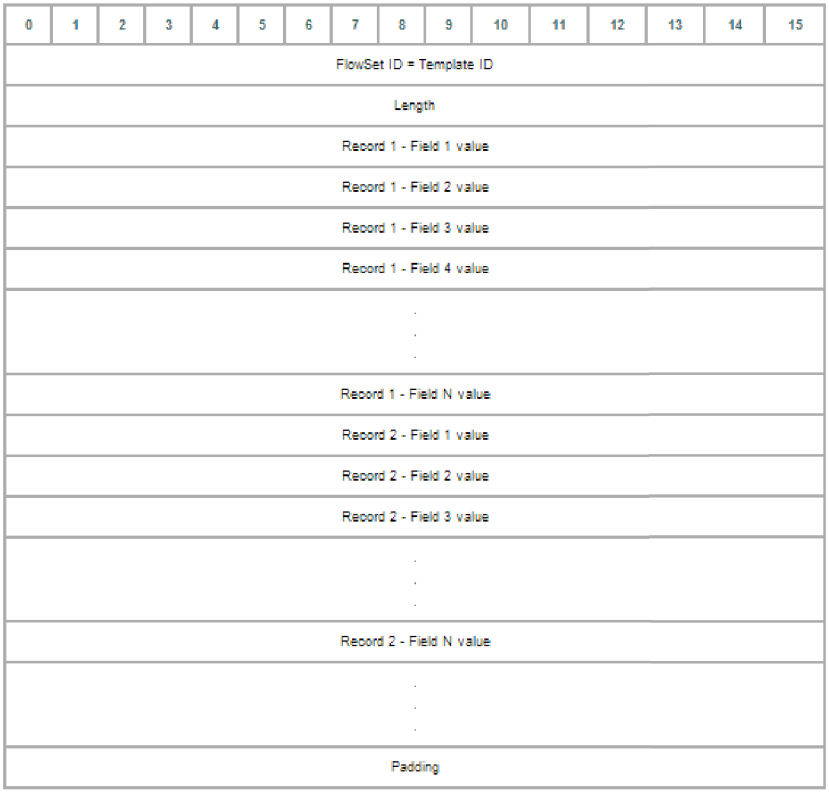


Figure 2.6: Scheme of NetFlow v9 data FlowSet. [[41]]

is at least the basic 5-tuple (Src. IP, Dst. IP, Src Port, Dst. Port, Protocol) necessary to distinguish a connection from all others. In flow sampling, for each unique combination that is present during a predefined time window different statistics are calculated.

These calculations are in most of the cases simple counting or adding. For example, the outgoing packets and the incoming packets are simply counted separately while the outgoing and incoming bytes are simply added up separately during the time frame. The flags that were present during this time window are aggregated by using a binary OR function, which simply means that a specific flag has either been present and will therefore be represented in one bit (0 or 1) independent of how often it has been present.

In the previous section, the term "time frame" or "time window" was mentioned several times. This time window is one of the essential parameters for a NetFlow sampling setup, because it defines how fine-grained the information will be or expressed in the opposite way how abstract the view on the current network traffic will be. This parameter defines the time span over which each connection will be sampled to a flow. In case a connection is active longer than the specified time span, the flow sampling component will export statistics calculated for this connection and start the computation again for this connection, but in a new flow. Basically this means that very short connections will result in one single flow and long (exceeding the time window) connections will end up in several flows.

Figure 2.7 illustrates the flow sampling concept based on time frames.

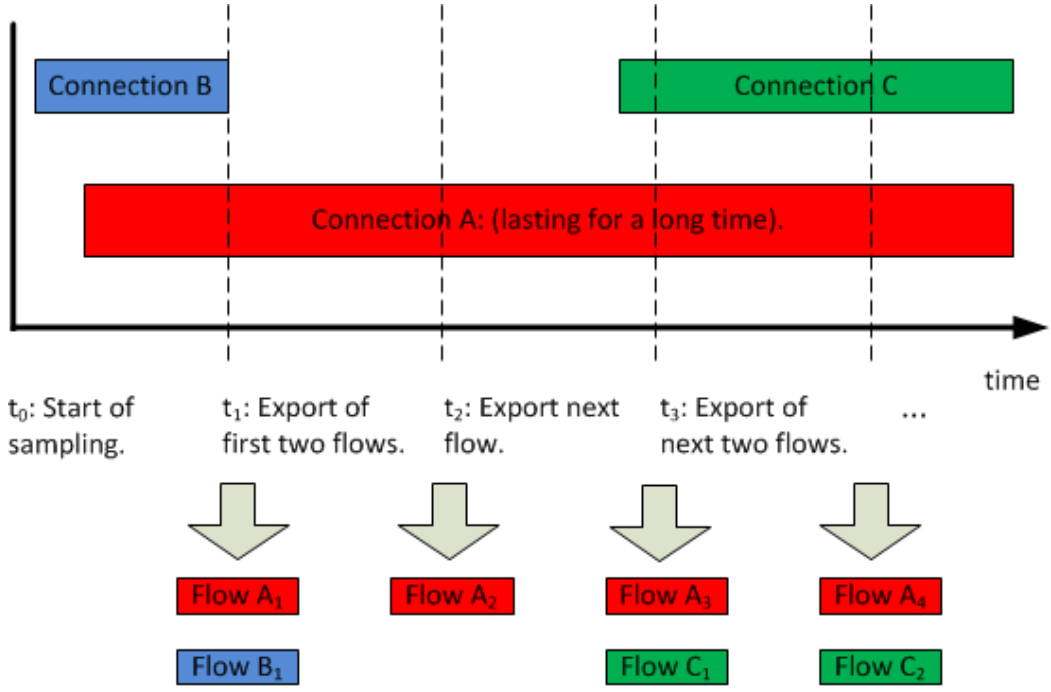


Figure 2.7: Concept of flow sampling based on time frames.

Bearing this concept in mind is important because, if the time frame is shorter, the probability to find flows that consist of only one packet with a certain TCP flag could be found increases. In contrast if the time window is longer the probability to end up in flows that have numerous TCP flags set increases.

None of these options is good or bad by itself, but depending on the use case and the envi-

ronment one might be more suitable then the other. In general, it is important to define what is more important for a particular use case. For the use case of detecting NAT it is probably better to have a more fine-grained view on the network, because a higher level aggregation can be done afterwards, if necessary. Hence this is not possible the other way around. In other use cases, space efficiency and therefore a more high level aggregation - by means of a longer time window - might be more important.

Furthermore, it is also important to take into account that besides the time based sampling algorithm there are also other sampling algorithms depending on the sampling device. For example, some sampling components might also support deterministic or random sampling. These methods and some of the routing devices implementing these methods are described in [[9]]

Another important thing to consider is that normally the flow cache is limited and manufacturers like CISCO normally use a technique called aging to expire older flows especially when the buffer gets filled up completely. This can cause a loss of some flows and the problem of missing data has to be considered in the later NAT detection approach.

The following section focuses on basic concepts related with exporting the sampled flows.

2.3.4 Flow Export and Collection

Exporting the flows to a so-called "Flow-Collector" is the next step in the chain of steps to detect NAT. This step in the chain is important because it could have a strong effect on the produced data set. The reason for this importance is that in this step there are at least two reasons why data can get lost. Because of this, the received data set for the actual detection component could be incomplete and flows could be missing.

The problem of missing values is important to consider in the detection component, because if, for example, this component performs a "Stateful Packet Inspection", such as analyzing the common three-way-handshake in TCP. It might happen, that parts of these handshakes are not present in the data set, even if the handshake was successfully performed in reality.

The reason why this loss of data is possible is because the Flow-Exporter transfers the sampled flows via UDP to the Flow-Collector and the UDP protocol is a simple message based and connectionless protocol, which means that it does not have any transmission control capabilities. This simple approach is used because it reduces the workload for ensuring the transmission, which would also make buffering of the transmitted flows necessary to be able to retransmit them in case they did not reach their desired destination.

The second reason why flows could get lost is because of limited buffer memory in the in the Flow-Exporter. This is because, regarding to Cisco, "for a Version 5 datagram, up to 30 flows can be sent in a single UDP datagram of approximately 1500 bytes" [[12]].

If the reliability of the NetFlow Export and Collection is important, then it might be a good idea to use this approach proposed by CISCO [[7]].

An example of the NetFlow version 9 Packet Format is provided by the following figure 2.8:

The NetFlow version 9 export format looks like follows [2.9]:

Now, that the foundations of NAT and NetFlow were described, the following chapter will provide the necessary foundations for classification and simulation.

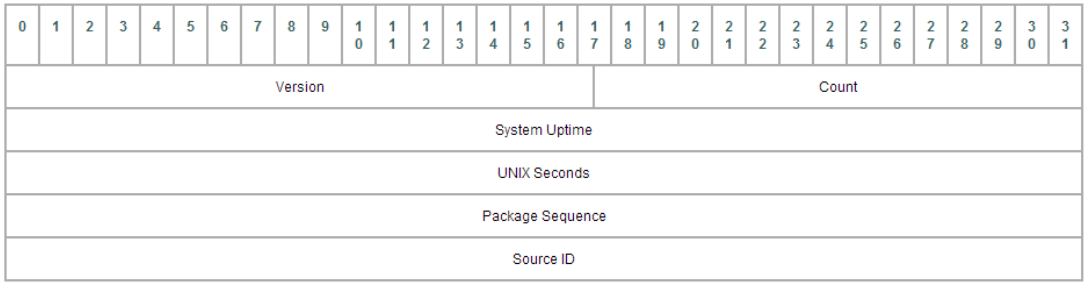


Figure 2.8: Packet Header of NetFlow version 9 [2.9]

Figure 2.9: NetFlow v9 Export Format [[41]]

CHAPTER 3

Foundations of Classification and Simulation

This chapter provides the fundamentals that are necessary for any simulation and classification task. Hence this chapter especially concentrates to the requirements of the NAT detection problem, which should be solved based on NetFlow data.

3.1 Fundamentals of Feature Creation and Selection

Any system that is intended to classify a set of samples into a number of different classes, needs some sort of characteristic trait(s) or symptom(s) that contain information about the class the sample belongs to. In the field of pattern recognition, such a characteristic trait is usually called a feature. The number of features that is needed to specify a sample object correctly depends on the use case and normally varies depending on the problem setting.

Building a classification system is usually based on two fundamental steps. The first step is to discover some unique trait of behaviour that can be used as a feature to determine the class a sample object belongs to. In practice, this normally leads to a set of features that contain information about the class a feature belongs to. Especially when the set of these features becomes very large, then it might be useful to just consider a subset of features, to make the decision process more efficient. Hence, the question which then arises is, which feature to choose and which to ignore. It is obvious that one would like to choose the optimal feature, which provides the most information or expressed in other words one would like to choose the feature that enables to classify the sample object with the highest certainty. In the fields of pattern recognition and machine learning this process is called feature selection or feature sub set selection.

The following sections provide more detailed fundamentals of this important aspect of classification problems.

3.1.1 Preliminaries

As described before, the motivation of feature selection is to select the optimal feature or feature set for a given classification problem. Hence depending on the actual classification task and other factors the definition of optimal might change. In fact, in practice there are numerous requirements a feature or feature set has to meet. Some of these requirements are in contrast to each other and therefore hard to fulfill at the same time.

Of course, besides some problem dependent requirements, there are also some more general requirements that each classification system has to meet somehow. Often finding the optimal feature (subset) is based on finding the best compromise between these contrary requirements. As described by H. T. Nguyen in [[25]], three basic requirements were figured out for the field of intrusion detection systems. These three requirements are "Reliability", "Efficiency" and "Effectiveness". Normally, a system will not be able to fully meet all of these requirements at the same time. The following figure 3.1 illustrates this principle.

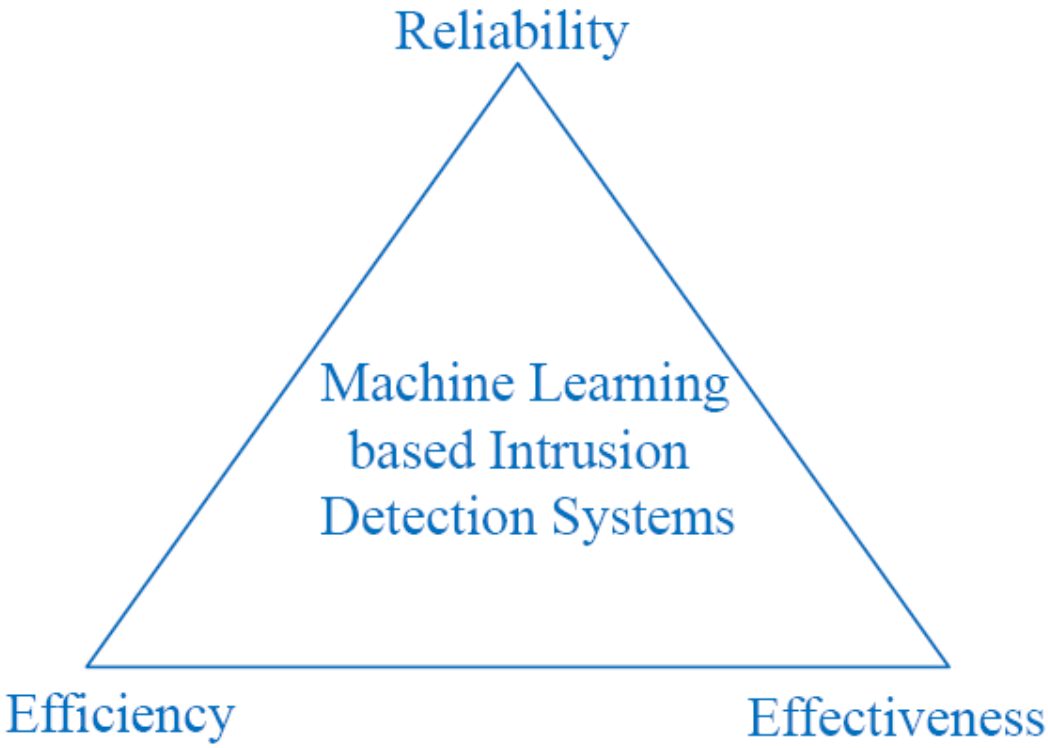


Figure 3.1: Requirements to Features and ML based Intrusion detection systems.[[25]]

Since the problem setting of this thesis is in a similar domain as in [[25]] these requirements will also apply for the features and classification approaches discovered and developed in later parts of this thesis.

Reliability in this domain means that the classification algorithm should be able to classify most of the provided sample instances correctly. In other words, this simply means that the classification algorithm should have a consistently high detection rate, even on slight changes in the data set. The requirement of course also applies to the feature (subset) since this is the basis for any classification algorithm.

Efficiency is of particular relevance in this field, since the data sets that have to be analyzed

in the field of network intrusion detection and/or Network Forensics are normally quite large. This means that such a system must be able to process even traffic volumes in the range of terabytes per day.

Effectiveness of a classifier in this domain means that the classifier should have high classification/detection accuracy. This basically means that the classifier should not produce too many falsely negative and falsely positive classified samples.

Since the selected features have a great impact on the later used algorithm, for classification, these requirements also apply to the features and not only the classification algorithms. Because of this, these requirements have to be kept in mind even at these early steps of feature creation and optimal feature selection for the later classification algorithm.

3.1.2 Different Techniques of Feature Selection

In general, the process of creating and selecting features can be done in two fundamentally different ways. The classical way for simple classification problems is to analyze and discover the relevant features for a classification system by hand. Hence, this only works for very simple problems. Many classification problems of today have a complexity, which cannot be handled by humans any more. Because of this, these problems have to be tackled in a different way. A great way to solve these very complex problems is to use machine learning techniques. Machine learning based classification is especially of interest in this field, because it makes it possible to also automate the feature recognition and selection phase of such a system. This especially supports the requirement of effectiveness.

In [[25], pp. 10] the three terms "Feature Creation", "Feature Extraction" and "Feature Selection" are distinguished in the following way. Feature Construction is described as the process of finding a subset of features that contains the most information but is as small as possible at the same time. In spite of, using machine learning methods for selecting features automatically, it is still necessary to distinguish between the two steps, feature extraction and selecting the best feature(s) from a set of created / discovered features.

Feature extraction is basically used as a synonym for any process of discovering informative attributes or input values that can be used to determine the class a sample object belongs to. This process can either be done manually, for example by defining virus signatures for antivirus programs, or it can also be done automatically and is then normally done based on statistic properties of the attributes of a training sample set. Today, some often used methods for this process are for example frequent episode extraction [[28]], n-gram [[14]] analysis or association rule learning [[1]]. Discovering these features is obviously the first and at the same time the most crucial step in the whole process of solving a classification problem. After having discovered features that can be used for the actual classification it might often be useful to select just the most relevant features from a set of features.

This step is then called feature (subset) selection. Even though it would be possible to perform the classification based on the complete feature set. Hence this has some disadvantages or even might not be possible in practice, because for example the set of features might be too large and complex. Therefore, the classification process would be to resource consuming. In fact, a proper feature selection can be helpful to reduce the data that has to be kept in the storage and also has to be processed by the algorithm later on.

For the optimal feature selection process several different techniques can also be used. Besides the manual selection, numerous methods for automatic feature selection were developed

in the past. Some often used methods today are for example the wrapper method, the filter model or embedded model. The main idea of the wrapper model is to use a learning algorithm based on different feature subsets. The performance of the algorithm is then used to determine, which of the feature subset selections has been the best one.

In contrast to this, the filter model is solely based on statistical characteristics of a data set and is not using any learning algorithm. The filter model is based on two different parts, namely feature ranking and feature subset evaluation. During the feature ranking, a rank or weight is assigned to each feature, based on the individual relevance to the target concept. In contrast to this, during the feature subset selection part of the filter model, features are, besides their rank, also selected depending on their correlation to each other. This can be useful, to avoid the selection of redundant features, which is especially likely to happen on high dimensional feature spaces. Besides this advantage, the filter model is also often applied on high dimensional feature spaces, because of its computational efficiency.

The embedded feature selection completely differs from the wrapper and filter method described before. In the embedded feature selection model, the process of selecting features is an integral part of the model building process. This is for example commonly done in the decision tree learning, where the decision at each branch is based on the selection of a feature.

Further details about the feature selection process can be found in [[4]] and [[16]].

3.2 Classes of NetFlow Related Features

In this section, features are distinguished by the source of their introduction. The sources are divided into the three categories: "(NAT) Gateway related", "Host related" and "User behaviour". These three classes will be described in the following sections in more detail.

3.2.1 Process related Features

Features introduced by the NAT Gateway are any changes that a Gateway does to the network packets in order to perform the process of doing the network address translation. Some of these changes are quite obvious in general and others reveal regularities or irregularities in the algorithms used by common NAT Gateways.

For example, it is obvious that the NAT-Gateway has to change the source IP addresses of outgoing and the destination addresses of incoming packets, by the definition of the NAT process. Other changes introduced by a NAT Gateway are not that obvious and not part of the definition of the process. For example, NAT-Gateways normally tend to decrement the time to live value (TTL) of a translated IP packet by one. In the related work chapter some other influences such as typical patterns in the IP-ID field have already been described.

Of course, since the NAT-Gateway has to perform some extra actions such as changing IP addresses of each packet and managing its translation table, this might potentially introduce some patterns in the time domain. Hence, assuming these pattern in the time domain really occur, then they can be assumed to be in a range of μ seconds or even nano seconds. Keeping in mind that only NetFlow data is considered in this thesis and that NetFlow is a statistical aggregation of network packets in the range of milliseconds and seconds this might not be a very promising approach on this data set.

3.2.2 User Behaviour related Features

User behaviour related features are features that are influenced by what the user does. In more detail, these features are influenced by the remote services consumed and the applications used for consuming them. Furthermore, the intensity, with which the user uses certain applications and/or services, heavily influences these features.

When it comes to NAT detection based on NetFlow, then there are some features that are obviously influenced by the user. For example the number of packets transferred in each direction (outgoing as well as incoming), the number of bytes transferred in both directions and the IP address of the remote service, which depends of course on the service chosen by the user.

Besides these obvious user behaviour related features, there are also other features influenced by the user that might not be that obvious. For example, the number of flags is influenced by the user in an indirect way, depending on how many connections and which type of connections the user creates. Furthermore, the used port numbers as well as the number of different ports used depends on how active the user is. The same holds for the used protocols and how actively each of these is used. Active in this context means, how many connections a user establishes and how many packets and bytes are transferred via these connections.

3.2.3 Host related Features

Besides the NAT-Gateway on the one end or the user on the other end, the host in-between these two sides can also introduce features that might be used for the NAT detection. Hence, the term host in this context is not precise enough, because the host actually consists of at least three parts. This is due to the fact that the host is based on the hardware, is running an operating system and in addition the actual applications. One might argue that the applications should better be classified under user behaviour class, but since this strongly depends on the actual application, this cannot be finally decided.

Features on the hardware level of the hosts might for example concern the clock deviations or changes in the response time due to changes in the processor load. However these features cannot be detected based on NetFlow and are therefore not considered relevant in this thesis.

On the operating system level, it already was shown that different operating systems can be distinguished by small changes in their TCP/IP Stack implementation. Since these features are for example related to the responsiveness, the usage of port numbers or the size of certain packets, this level is worth further analysis.

On the application level it was also shown that running instances of various applications can be remotely detected. Hence this is highly dependent on the application and a very complex problem to solve on such high level data such as NetFlow. Hence, some types of applications on this level might be of particular interest for detecting NAT. For example, Voice over IP applications such as Skype and GoogleTalk or Cloud Storage Services such as DropBox, Google Drive or Skydrive only allow one running instance per user per Host at a time. Detecting several active instances of one of these applications on one particular IP address also means that multiple hosts/users are active. This also means that these hosts obviously share the same internet line.

3.3 Fundamentals of classifiers

After the extraction of features was described in the last section, this section provides an overview of different methods how these features can be used for classifying the flows into the classes "NATed" or "not-NATed".

3.3.1 Preliminaries

A classifier is a method to automatically make decisions about a data set based on one feature or a set of features. Regarding to [[23]], in mathematical terms, a classifier is a function f that maps input feature vectors $x \in X$ to class labels $y \in \{1, .., C\}$.

Numerous methods exist to create a classifier, but there are two fundamentally different types of classifiers, depending on the way they work. One possible type is to build classifiers based domain specific expert knowledge. The other possible type is to use some sort of artificial intelligence and machine learning methods.

There are some fundamental requirements a classifier has to meet. For example, it is essential that the classifier has certain reliability. In this context, reliability means that the classifier has to be able to make a certain degree of right decisions. This degree of right classification is also often called detection performance. This detection performance should also be at a certain level under slightly changing conditions.

The detection performance could only be measured in case labeled training data is available. That means that the classifier is applied on a certain amount of data samples that are known to be positive or negative examples. Then the results (labels output by the classifier) of the classification are compared with the labels from before to measure in how many cases the classifier came to the right result and in how many cases it failed.

As described before, there are two fundamentally different ways on how to build classifiers.

The first way is to use domain specific expert knowledge about certain features. As the name of this method already indicates, this method is mainly based on knowledge about specific behaviour of the system - in case of NAT detection it is knowledge of the process of performing NAT and especially about unique patterns introduced by different devices. Later in this thesis two classifiers that rely on this method will be presented.

The second way to create classifiers is to use machine learning techniques. This is mainly based on the creation of training data samples. The classifier could then be built by either a supervised or unsupervised learning method. Unsupervised learning of the classification model is done by providing an unlabeled set of training data. Then for example a clustering algorithm separates the supplied samples into a number of different classes. Hence, this process does not involve any assignment of labels. This means, no information is given which of the clusters is belonging to which class of traffic. Therefore, a further step of assigning labels to these clusters would be necessary. This kind of approach was for example already often applied in the field of general anomaly detection systems. Nevertheless, the problem setting in this thesis is not to detect some sort of anomaly, but to label a given input sample. Therefore, supervised learning is the more natural choice for this problem.

In contrast to this, supervised learning is performed by providing a set of labeled data samples, which are then used to train the classifier(s). In supervised learning, the underlying model is inferred based on these training data sets automatically. In the case of NAT detection,

this means that flow samples from IP(s) that are known to perform NAT have to be extracted as well as flow samples that come from a source that is known to be not NATed.

There are several important requirements, which have to be met by such a classifier. In the domain of intrusion detection systems, the classifier's detection rate is normally determined as the number of attacks (true positives - TP) divided by the total number of attacks (Number of Positives Samples - S_p). Formally this is denoted as:

$$DetectionRate = \frac{\#TP}{\#S_p} \times 100 \tag{3.1}$$

The equation above delivers the detection rate per cent. In the context of this thesis, the true samples do not necessarily have to be interpreted as attacks. Hence, when it comes to the use case of detecting unauthorized Wi-Fi routers within an enterprise network, it would also be right to interpret these samples as an attack or intrusion attempt against the network.

Within this thesis, the detection rate is defined as the number of all correctly classified samples. This means that it will be based on the True Positive (TP) and True Negative (TN) Rate in relation to the total number of samples. This can be denoted formally as follows:

$$DetectionRate = \frac{TP + TN}{\#Samplesintotal} \times 100 \tag{3.2}$$

As before, this equation delivers the detection rate per cent. The adaption in the definition of the detection rate is necessary, since the detection problem in general is to determine whether NAT is used within a network or not. This slightly differs from the definition used in intrusion detection systems, where the attacks (TP) are much more relevant than the negative case (TN).

Nevertheless, besides the detection rate, a classifier also has to meet other important requirements such as for example generalization or runtime performance. Generalization performance is the capability of a classifier, trained on a limited training set, to operate satisfactory on unkwon/unseen new samples the samples that have not been present during the training phase. Furthermore, generalization is also related to the problem of missing samples or incomplete data. It is obvious that the runtime performance is also a very important trait of a classifier.

According to [[44]], the generalization capability of a machine learning algorithm can be predicted. This prediction is based on the so-called Vapnik-Chernovenkis (VC) dimension, which is basically a probabilistic prediction of the upper bound of the test error of a classification model. We assume that a machine learning algorithm f as well as the function $TrainingError(f)$, which is providing the training error, are given. We further assume it is assumed that all the training samples and testing samples are selected from exactly the same generated distribution. If under these conditions both assumptions hold true, then the bound on the test error of a model can be determined, with a probability of 1 - p, in the following way:

$$TestError(f) \leq TrainingError(f) + \sqrt{\frac{h\left(log\left(2m/h\right) + 1 - log\left(p/4\right)\right)}{m}} \tag{3.3}$$

In the equation above, m is the number of maximum training samples and h is the VC dimension, which is equal to the power of f. This bound can be used to estimate a classifier's

error on unknown sample objects on which it will be applied in the future, solely based on the training error measurement and the VC dimension of f. Hence, the above assumption of having exactly the same distribution on the training as well as the test set is not very realistic and often does not hold true in practice. This is obvious, since in practice the extraction of these two data sets is split into distinct processes, and therefore the generation of the distribution is different on both data sets. Because of this limited applicability, at least in the problem domain of network intrusion or NAT detection, one has to consider other methods of estimating the generalization performance of a classification algorithm, such as for example the off-training set (OTS) generalization error.

Since a classifier should also be able to operate in real-time in ISP or enterprise networks, it has to be able to deal with a huge amount of data in a very short amount of time. Therefore, besides the generalization of course also the computational complexity is of importance since this highly affects the runtime performance.

Some commonly used classification techniques are for example Decision Trees, Neural Networks or Support Vector Machines, among many others. Since these techniques are used later within this thesis, a more detailed theoretical description is provided in the following sections.

3.3.2 Support Vector Machines

In the previous section it was mentioned that classification tasks can generally be grouped into two classes, namely linear and nonlinear classification tasks. Support Vector Machines are capable of operating in both classes.

A Support Vector Machine (SVM) is a supervised learning model that could be applied for linear as well as non-linear classification tasks.

In the most basic case, the SVM is used as a binary classifier that maps each sample of a given sample set to one of two classes. If these classes are linearly separable, then the SVM learning task is to learn a hyperplane with a maximum margin to both of the classes. This maximum margin is necessary to allow either class to "move" a little bit. This makes the model more robust against misclassification in case samples occur near the class boundaries that have not been present at training time of the model.

In general, an SVM constructs a hyperplane or a set of hyperplanes in a high or even infinite dimensional feature space. Even though a classification problem might be defined in a finite-dimensional space, it might be solvable by transforming it into a higher dimensional space. Hyperplanes are always defined by their direction and exact position in space. The following formal description of support vector machines mainly follows the descriptions provided in [[11]] and [[34]].

In the linear case, it is assumed that the data samples that have to be assigned to a label are linearly separable. Expressed in other words, this simply means that it is possible to draw a line on a two dimensional grid, that separates the two classes from each other. According to [[11]], this holds true under the assumption that we have L training points in such a case and that each input x_i has D attributes (i.e. is of dimensionality D) and is in one of two classes $y_i = -1 \text{or} +1$. Further, assuming the training data is of the form x_i, y_i where $i = 1...L, y_i \in -1, +1, x \in R^D$.

As described before, in the two dimensional case a line based on the graph x_1 vs. x_2 is then enough to separate the two classes in case $D = 2$. In the case of $D > 2$ a hyperplane on graphs of $x_1, x_2...x_D$ is used instead to separate the two classes [[11]].

Such a hyperplane is denominated mathematically by the formula:

$$w \cdot x + b = 0 \tag{3.4}$$

In this formula:

- w is normal to the hyperlane.
- $\frac{b}{||w||}$ is the perpendicular distance from the hyperplane to the origin.

Support Vectors are the data points of the sample data set which are closest to the separating hyperplane, under the condition that they are as far as possible away from the closest members of both classes. This principle is illustrated in Figure 3.2.

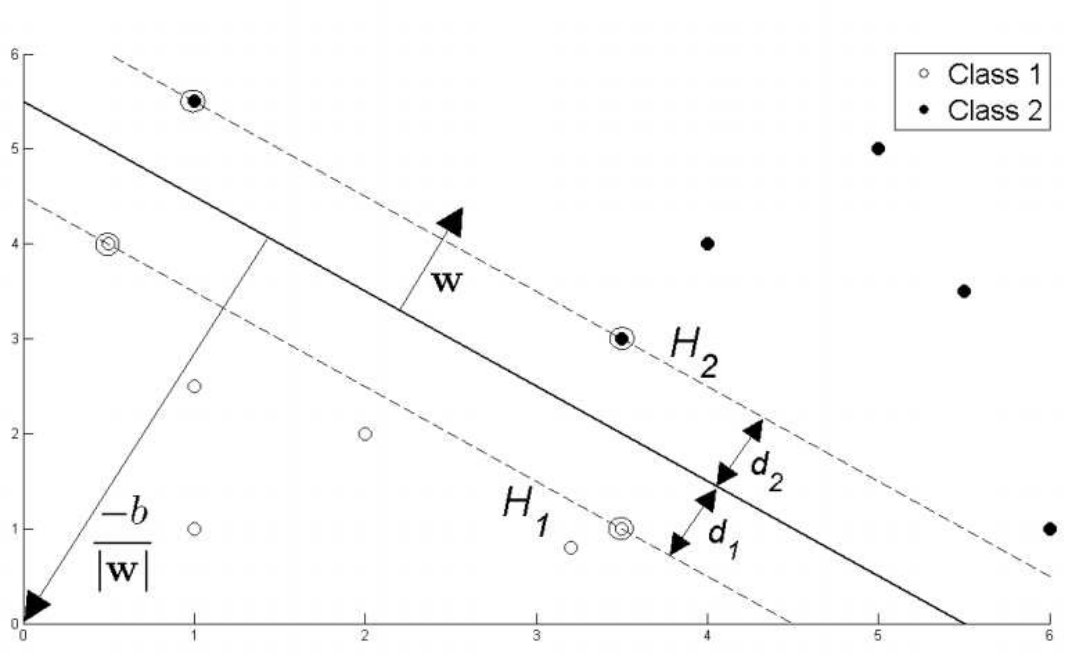


Figure 3.2: Principle of linear classification using SVM [[11]].

The figure 3.2 also illustrates that creating an SVM mainly depends on choosing the variables w and b in a way that the training data set can be described by the following two equations:

$$x_i + b \geq +1 \text{ for } y_i = +1 \tag{3.5}$$

$$x_i + b \leq -1 \text{ for } y_i = -1 \tag{3.6}$$

According to [[34]], these two equations can be combined, which results in the following equation:

$$y_i \left(x_i \cdot w + b \right) - 1 \geq 0, \forall i \tag{3.7}$$

In case only the points that are closest to the separating hyperplane, such as for example the Support Vectors, are considered then the two planes H_1 and H_2 can be described by the following two equations:

$$x_i \cdot w + b = +1 \text{ for } H_1 \tag{3.8}$$

$$x_i \cdot w + b = -1 \text{ for } H_2 \tag{3.9}$$

Each of the two planes has a certain distance to the hyperplane. As illustrated in Figure 3.2, the distance d_1 describes the distance between H_1 and the hyperplane while d_2 describes the distance between H_2 and the hyperplane. Based on these two distances the so called margin of an SVM is calculated. The margin describes the equidistance of the hyperplane from H_1 and H_2.

The fundamental goal of SVM classification is to direct the hyperplane in a way that it is as far away from the Support Vectors as possible. Therefore building an SVM classifier can also be seen as optimization problem. The target of such an optimization problem is to creating the maximum margin. Constructing an SVM classifier based on such this margin is important in order to avoid, having too many misclassified samples, which are close to the borders of the two classes. Therefore choosing the maximum margin ensures that the separating hyperplane is not too close to one of the two classes, which then reduces the chance of misclassification.

Regarding to [[11]], "simple vector geometry shows that the margin is equal to $\frac{1}{||w||}$ and maximizing it subject to the constraint is equivalent to finding:"

$$min||w|| \text{ such that } y_i \left(x_i \cdot w + b \right) - 1 \geq 0, \forall i \tag{3.10}$$

Furthermore, it is also important to mention that the minimization of $||w||$ is equivalent to minimizing $\frac{1}{2}||w||^2$. This transformation of the problem makes it possible to use Quadratic Programming (QP) optimization later. This is expressed by the following equation:

$$min\frac{1}{2}||w||^2 \text{ such that } y_i \left(x_i \bullet w + b \right) - 1 \geq 0 \forall i \tag{3.11}$$

To meet the constraints of the minimization, it is necessary to introduce Lagrange multipliers α, with $\alpha_i \geq 0$, $\forall_i$. This is shown in the following equations:

$$L_P \equiv \frac{1}{2}||w||^2 - \alpha \left[y_i \left(x_i \bullet w + b \right) - 1, \forall_i \right] \tag{3.12}$$

$$\equiv \frac{1}{2}||w||^2 - \sum_{i=1}^{L} \alpha_i \left[y_i \left(x_i \bullet w + b \right) - 1 \right] \tag{3.13}$$

$$\equiv \frac{1}{2}||w||^2 - \sum_{i=1}^{L} \alpha_i y_i \left(x_i \cdot w + b \right) + \sum_{i=1}^{L} \alpha_i \tag{3.14}$$

The goal is to find the combination of w and b which minimizes, and the α which maximizes equation (3.14). It is important to keep in mind that the constraint $\alpha_i \geq 0 \forall_i$ should be fulfilled at the same time. This can be done by differentiating L_P with respect to w and b and by setting the derivatives to zero:

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{L} a_i y_i x_i \tag{3.15}$$

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow \sum_{i=1}^{L} \alpha_i y_i = 0 \tag{3.16}$$

By substituting (3.10) (3.11) into (3.9) a new formulation can be retrieved, which needs to be minimized depending on α:

$$L_D \equiv \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \text{ such that } \alpha_i \geq 0 \forall_i, \sum_{i=0}^{L} \alpha_i y_i = 0 \tag{3.17}$$

$$\equiv \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i H_{ij} \alpha_j \text{ where } H_{ij} \equiv y_i y_j x_i \cdot x_j \tag{3.18}$$

$$\equiv \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha^T H \alpha \text{ such that } \alpha_i \geq 0 \forall_i, \sum_{i=1}^{L} \alpha_i y_i = 0 \tag{3.19}$$

According to [[11]], this new formulation of L_D is also known as dual form of the primary L_P. The dual form only requires the dot product of each input vector x_i to be calculated, which is important for the kernel trick that will be described later.

The advantage of the reformulation is that it transformed the optimization problem of minimizing L_P into maximizing L_D. This means that now problem can be expressed in the following way:

$$\max_{\alpha} \left[\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \alpha^T H \alpha \right] \text{ such that } \alpha_i \geq 0 \forall_i \text{ and } \sum_{i=1}^{L} \alpha_i y_i = 0 \tag{3.20}$$

According to [[33]], this problem is a convex quadratic optimization problem. Such quadratic optimization problems can be solved by using a quadratic problem solver, which then obtains the values for α and w. Then the only remaining part that still has to be calculated is b.

Any data point that is satisfying formula (3.11) and which is also a Support Vector x_s will then have the following form:

$$y_s(x_s \cdot w + b) = 1 \tag{3.21}$$

When substituted into (3.10):

$$y_s \left(\sum_{m \in S} \alpha_m y_m x_m \cdot x_s + b \right) = 1 \tag{3.22}$$

Where S denotes the set of indices of the Support Vectors. For determining S, the indices i have to be found according to the condition $\alpha_i > 0$. Sub sequentially, it has to be multiplied through by y_s and then $y_s^2 = 1$ used from (1.1)and (1.2):

$$y_s^2 \left(\sum_{m \in S} \alpha_m y_m x_m \cdot x_s + b \right) = y_s \tag{3.23}$$

$$b = y_s - \sum_{m \in S} \alpha_m y_m x_m \cdot x_s \tag{3.24}$$

Instead of using an arbitrary Support Vector x_s, it is better to take an average over all the Support Vectors in S:

$$b = \frac{1}{N_s} \sum_{s \in S} (y_s - \sum m \in S \alpha_m y_m x_m \cdot x_s) \tag{3.25}$$

Now the variables w and b, which define the separating hyperplanes optimal orientation, were retrieved.

In case the classes are not completely linearly separable, the setup described above is no longer valid. This is also illustrated in figure 3.3.

In such a case the constraints (3.1) and (3.2) are loosen a bit to also allow the misclassification of feature vectors. This is done by introducing a positive slack variable $\xi_i, i = 1, ..., L$ [[11]]:

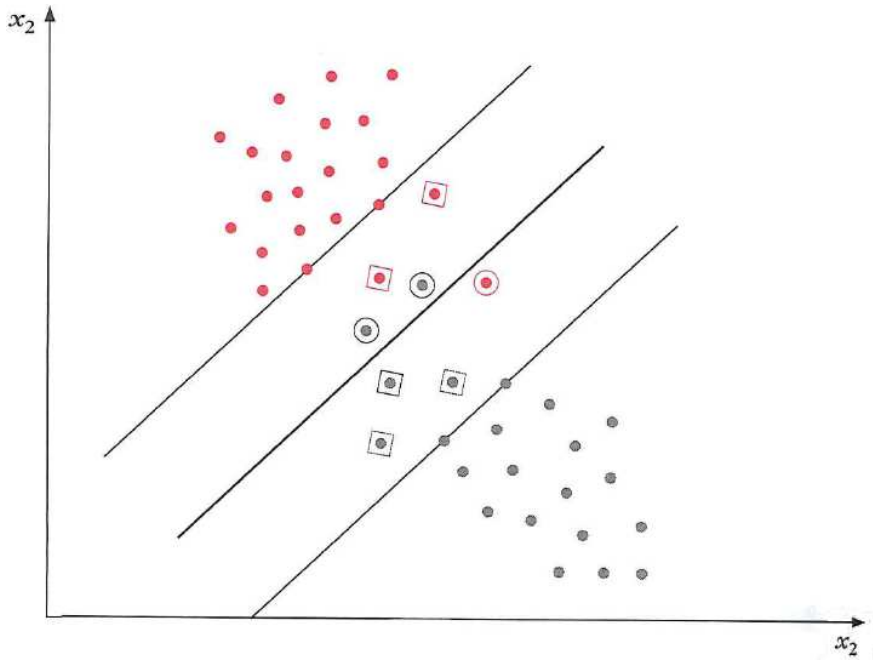


Figure 3.3: Non separable Class Example. [[34]]

$$x_i \cdot w + b \geq +1 - \xi_i \text{ for } y_i = +1 \tag{3.26}$$

$$x_i \cdot w + b \geq -1 + \xi_i \text{ for } y_i = -1 \tag{3.27}$$

$$\xi_i \geq 0 \forall_i \tag{3.28}$$

All these equations can be combined into the following single equation:

$$y_i(x_i \cdot \bullet w + b) - 1 + \xi_i \geq 0 \text{ where } \xi_i \geq 0, \forall_i \tag{3.29}$$

As shown in figure 3.4 a penalty is assigned to those data points that are on the wrong side of the margin boundary.

This penalty is increased the larger the distance to the margin boundary is. In general the goal is to reduce the number of misclassifications. A way to do this is to adapt the target function (3.19) to:

$$min \frac{1}{2}||w||^2 + C \sum_{i=1}^{L} \xi_i \text{ such that } y_i \left(x_i \cdot w + b\right) - 1 + \xi_i \geq 0 \forall_i \tag{3.30}$$

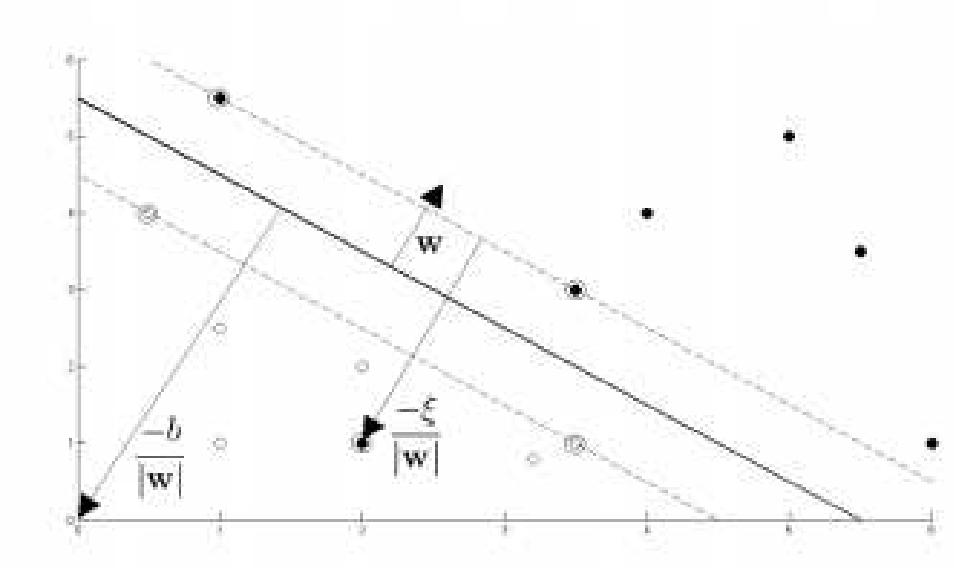


Figure 3.4: Hyperplane of soft margin SVM.[[34]]

The parameter C in the equation above controls the trade-off between the slack variable penalty and the size of the margin. As done before, the next step is now to reformulate it as a Lagrangian, because the goal is to minimize with respect to the variables w, b and ξ_i and maximize with respect to α (where $\alpha_i \geq 0, \mu \geq 0 \forall_i$):

$$L_P \equiv \frac{1}{2}||w||^2 + C\sum_{i=1}^{L}\xi_i - \sum_{i=1}^{L}\alpha_i\left[y_i\left(x_i \cdot w + b\right) - 1 + \xi_i\right] - \sum_{i=1}^{L}\mu_i\xi_i \tag{3.31}$$

In the next step, it is necessary to differentiate with respect to w, b and ξ_i and setting the derivatives to zero [[34], p.126]:

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{L}\alpha_i y_i x_i \tag{3.32}$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1} L\alpha_i y_i = 0 \tag{3.33}$$

$$\frac{\partial L_P}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \mu_i \tag{3.34}$$

According to [[11]], in the next step these equations are substituted. LD then has the same form as (3.19) before. Nevertheless, formula (3.16) together with $\mu_i \geq 0 \forall_i$ implies that $\alpha \leq C$. It is necessary to find:

$$\max_{\alpha} \left[\sum_{i=1}^{L} \alpha_i - \frac{1}{2} \alpha^T H \alpha \right] \text{ such that } 0 \leq \alpha_i \leq C \forall_i \text{ and } \sum_{i=1}^{L} \alpha_i y_i = 0 \tag{3.35}$$

Like shown in the linear separable case before, b is calculated in the same way as in (3.6). Hence in this case the set of Support Vectors used to calculate b is determined by finding the indices i where $0 < \alpha_i \leq C$.

Until this point support vector machines were considered to separate two classes linearly by "drawing" a hyperplane. Hence, with a method, which is called the kernel method or also often called "the kernel trick", it is also possible to classify a data set in a non-linear fashion.

In the be beginning of this section one of the first steps was to create a Matrix H from the dot product of the input variables, as given below:

$$H_{i,j} = y_i y_j k(x_i, x_j) = x_i \cdot x_j = x_i^T x_j \tag{3.36}$$

In this equation $k(x_i, x_j)$ is an example of a Kernel Function. In particular $k(x_i, x_j = x_i^T x_j)$ is known as a linear Kernel. All kernels have in common that they calculate inner products of two vectors. If the function can be recast into a space of higher dimensionality, then only the inner products of the mapped inputs in the feature space need to be determined. This transformation is done by a so called feature mapping function $x \mapsto \phi(x)$. The main advantage of this method is that ϕ does not need to be calculated. [[11]][[34]]

Since many real world classification problems are solvable by using a linear separation method, using this "trick" helps to transform even classification problems of a very high dimensional space into a form where they become solvable again. The following figure [3.5] illustrates this principle.

Such a kernel can be defined in multiple ways. For example the kernel function can be defined to be based on linear function or a "Radial Basis Function (RBF)". An RBF Kernel function is defined in the following way:

$$k(x_i, x_j) = e^{-\left(\frac{||x_i - x_j||^2}{2\sigma^2}\right)} \tag{3.37}$$

Another often used kernel is for example the Polynomial Kernel:

$$k(x_i, x_j) = (x_i \cdot x_j + a) \tag{3.38}$$

Furthermore also the "Sigmoidal Kernel" is used quite often:

$$k(x_i, x_j) = tanh(a x_i \cdot x_j - b) \tag{3.39}$$

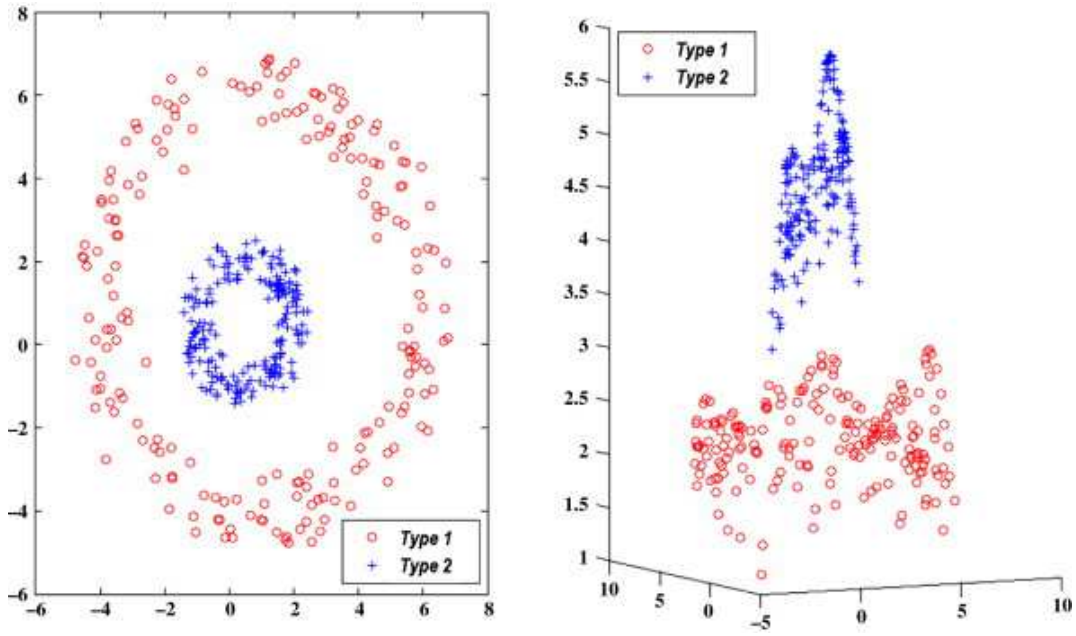


Figure 3.5: SVM 2D to 3D conversion and RBF Kernel. [[38]]

Of course this list is not complete and there exist numerous kernels in addition to those, that were presented before.

The advantages of using SVMs for classification are quite obvious. Since it tries to find the maximum margin between the classes while having minimum error in the classification, this method provides a good trade-off between precision and robustness when it comes to unseen data samples that are lying near the class borders. Furthermore, as mentioned before, it makes it easy to apply the kernel trick and therefore to deal with high dimensional classification problems.

Hence the major limitation of support vector machines is the high computational effort required during both the training and the test phase. This is due to the fact that support vector machines as shown before rely on Quadratic Problem solving. Regarding to [[34], p.129] "A naïve implementation of a quadratic programming (QP) solver takes $O\left(N^3\right)$ operations, and its memory requirements are of the order $O\left(N^2\right)$".

3.3.3 Artificial Neural Networks

Neural Networks are an artificial representation of the (human) brain that tries to simulate the learning process based on provided examples. In the figurative sense, these artificial neuronal networks are used to represent the process of human learning from experience.

Like a human brain, such a network consists of different nodes that are called neurons. In biology a neuron consists of several important parts. Figure 3.6 illustrates the biological model of a neuron.

From the figure 3.6 it can be seen that each neuron has a core on the inside that is called nucleus in biology. In the outside around the main part of its cell body it has numerous dendrites on which it takes input stimulations. On the opposing side it has a part that is called Axon. Via this axon output stimulations are transferred from the neuron to other neurons. This axon also has a trigger area which connects the axon to the rest of the neuron's cell body.

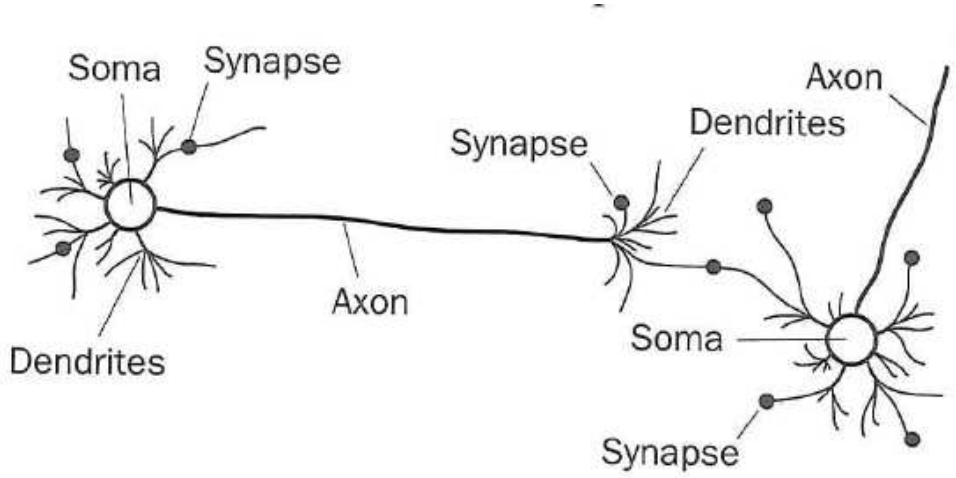


Figure 3.6: Biological Model of a Neuron.[[24]]

This trigger area influences the strength of the stimulation that is propagated further to other neurons. At the end of the axon, the neuron has numerous synapses to connect to other neuron dendrites.

An artificial neuron tries to mimic this concept in an abstract formal way by expressing this behaviour via a mathematical function.

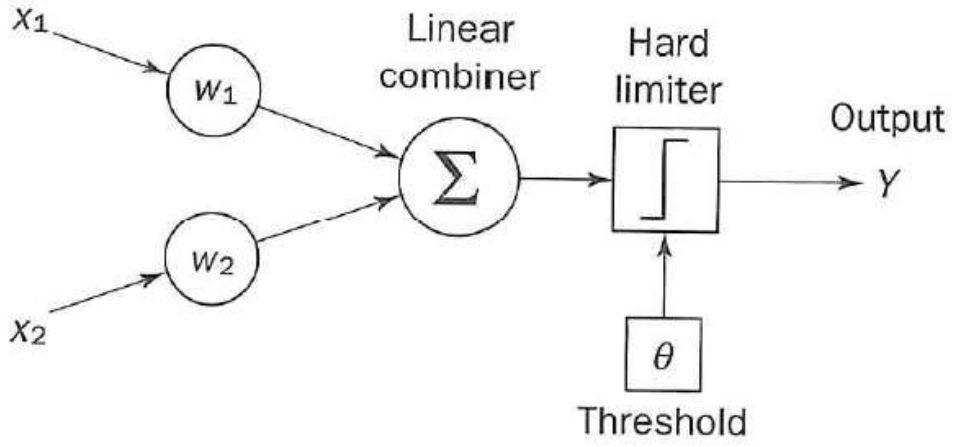


Figure 3.7: Model of an artificial neuron. [[24]]

The following table from M. Negnevitsky [[24], p.168] presents the analogy between the biological and artificial neuron model:

Each neuron of a neuronal network receives one or more inputs. The biological term for

Biological neural network	**Artificial neural network**
Soma	Neuron
Dentride	Input
Axon	Output
Synapse	Weight

Table 3.1: Analogy between the biological and artificial neuron model.

these inputs is dendrites. In general, these input signals are summed up to produce an output. The different inputs of an neuron are usually weighted. In biology, this output is called an axon. This summation is done non-linearly and passed through a so called activation function. This activation function determines the value put out by the axon. In most cases this activation function has a sigmoid shape, but others are also possible.

The synapses of a neuron, which is a part of a network, are then linked to the dendrites of the neurons in the next layer. This link is also called synapse in biology.

Many practical applications today are based on the McCulloch-Pitts-Neuron [[21]]. In this concept, the output of the activation function has to exceed a certain threshold value before the neuron outputs a logical one at its output.

Such a McCulloch-Pits-Neuron can be mathematically formulated in the following way:

$$y_k = \varphi\left(\sum j = 0 m w_{kj} x_j\right) \tag{3.40}$$

Neural networks are usually used in four steps:

1. Construction of the neuronal network and determining the neurons' activation function.
2. Train the network based on a training data set.
3. Testing the detection rate of the network based on provided test data.
4. Classification of the actual data set.

Training is the most important step of these four steps, because in this step the neuronal network learns the model for the later classification.

In theory, such a network has the following four opportunities to learn [[24]]:

- Adapting the weights of the inputs of a neuron.
- Establishing new connections or deleting existing connections
- Changing the threshold values of the neuron(s)
- Adding new and removing existing neurons.

Furthermore, changes in the activation function have a great influence on the network's behaviour. In practice, a sigmoid activation function is commonly used as kind of a standard approach. It is important to mention that this function is normally only adapted during the initial construction of the network, but not by the network itself. Hence any change of this function has to be done carefully and one should have a "good/special" reason to change it. In practice, normally the learning process of a neuronal network is based on adapting the weights

of the connections between its neurons.

A neuron model that is used very often in practice today is the Multilayer Perceptron. More details about this model will be given in the rest of this section.

Multilayer Perceptron

A multilayer perceptron is a feedforward implementation of a neural network that maps a set of input variables to a set of output variables. This model represents the simplest implementation of a neural network and was introduced by F. Rosenblatt 1958 [[31]]

Normally, such a network consists of one input layer, an output layer and at least one or more layers in between these two layers. Despite of the input nodes, all other nodes are neurons with non-linear activation functions. The multilayer perceptron belongs to the class of supervised learning techniques, because it makes use of the so-called backpropagation method during its training phase. Bearing this in mind then it is obvious that a multilayer perceptron always consists of at least three layers. As with most other neural networks, each of the nodes of one layer of the network connect to all other nodes in the following layer with certain weight. To learn a model the MLP has to adapt these weights. These adaptions are made piecewise for each sample of the training set. The correction of the weights is based on the amount of error occurring at the output layer.

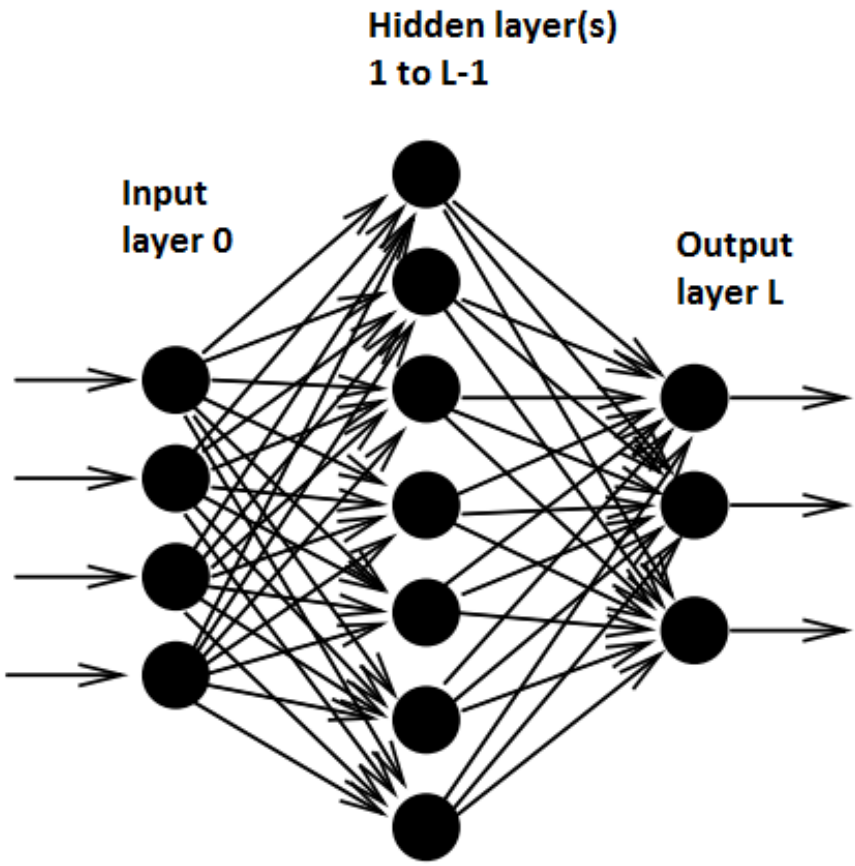


Figure 3.8: Model of Multilyer Perceptron.

This correction of the weights depending on the amount of error at the output layer is also

called **backpropagation**. This concept only exists with artificial neural networks, since real neural networks such as those from a real brain do not seem to work like this [[5]]. Regarding [[36]] the algorithm consists of four steps, which will be described in detail in the following paragraph.

The basic idea of this concept is to have a teacher during the training phase who knows exactly the corresponding output for a given input, at any time during the training. Therefore it is obvious that this concept belongs to the class of supervised learning strategies/methods.

The multilayer perceptron tries to adapt the weights of its internal nodes in a way that it reaches a target function as precisely as possible. Expressed in a different way, the goal of this concept is to minimize the deviation/error between the output of the neural network and the desired target output.

In general, the algorithm is based on four steps. An informal sketch of the algorithm might look like the following:

1. Input vector is fed to the input layer of the (neural) network.
2. Comparison of current output of the network with the desired output.
3. Backpropagation of the error from the output layer in direction of the input layer.
4. Adapting weights between the nodes(neurons) depend on their influence on the error.

Even though this informal description properly outlines the fundamental idea of the algorithm, it is not very precise on how these steps are put into practice. Therefore in the following, a more detailed formal description of the algorithm will be provided. This theoretical description is based on those provided in [[36]] and [[33]]. The description of the four steps of the backpropagation algorithm are based on the following simplified model and setting:

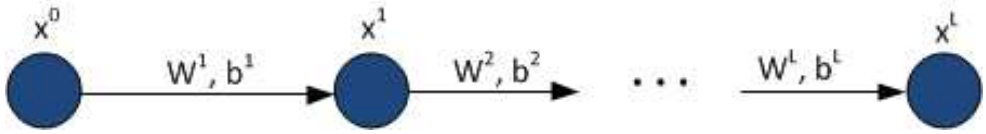


Figure 3.9: Simplified Multilayer Perceptron network.

In the simplified model that is shown in figure [3.9] and the following detailed description the following is assumed to hold true: $x^l \in R^{nt}$ for all $l = 0, ..., L$. Furthermore, it is important to point out, that - as described in the model above - the network consists of $L + 1$ layers of neurons and has L layers of synaptic weights. The aim of the backpropagation algorithm is to change the weights W and biases b in a way that the output x^L changes towards the expected output d.

The actual steps of the backpropagation algorithm are the following four [[36]]:

1. Forward pass. Transformation of the input vector x^0 into output vector x^L, by solving the following equation:

$$x_i^l = f\left(u_i^l\right) = f\left(\sum_{j=1}^{n_{l-1}} W_{ij}^l x_j^{l-1} + b_i^l\right) \tag{3.41}$$

for l= 1 to L.

2. Error computation. Describes the differences between the expected output d and actual outpud x^L. This is calculated by evaluating the following equation:

$$\delta_i^L = f'\left(u_i^L\right)\left(d_i - x_i^L\right) \tag{3.42}$$

3. Backward pass. In this step the error from the output layer is propagated backwards through the entire network. This backward propagation can formally be denoted as follows:

$$\delta_j^{l-1} = f'(u_j^{l-1}) \sum_{i=1}^{n_1} \delta_i^l W_{ij}^l \tag{3.43}$$

   from l = L to 1.

4. Learning updates. In this step the synaptic weights have to be updates based on the results from the forwarded and backward passes before. This means that the following two equations have to be solved:

$$\Delta W_{ij}^l = \eta \delta_i^l x_j^{l-1} \Delta b_i^l \tag{3.44}$$

   Both terms are evaluated for l = 1 to L.

Besides the backpropagation, also the concept of hidden layers is very important in the context the multilayer perceptron. It is necessary/a good idea to introduce at least one hidden layer, because the input layer mainly just accepts the input to the network, but hardly does any computation. Similarly the output layer just accepts a stimulation from some "lower" layer. Because of this the hidden layer(s) contain the computation and the learned model for the later classification. Regarding [[24], p. 174] single hidden layer is capable to represent any continuous function. A second hidden layer is even capable of representing discontinuous functions.

One crucial factor to influence a neural network is to change its size. This has to be done very carefully, since too small networks might not be able to classify the samples at all, but to large networks might tend to overfit to the training data set and is learning differences in the samples of the same class, what is not intended. Thus it is important to find the right balance between a large enough network that is able to learn the factors that make the samples similar, but is still small enough to build a general enough model that also fits unseen data samples.

Besides the generalization, the size of a network also highly influences its computational performance. Small networks are computationally faster and also "cheaper" to build. According to [[34]] in practice, the appropriate size of a neural network can be determined by using for example "Algebraic Estimation of the Number of Free parameters", "Pruning Techniques" or "Constructive techniques".

3.3.4 Decision Trees

One of the most intuitive concepts of classification is probably the decision tree based decision making. Regarding [[33]] these are also known under the more general term multistage decision system. In such a system classes are sequentially rejected until a class is reached that is finally accepted. The following description of decision tree learning is mainly based on [[33]].

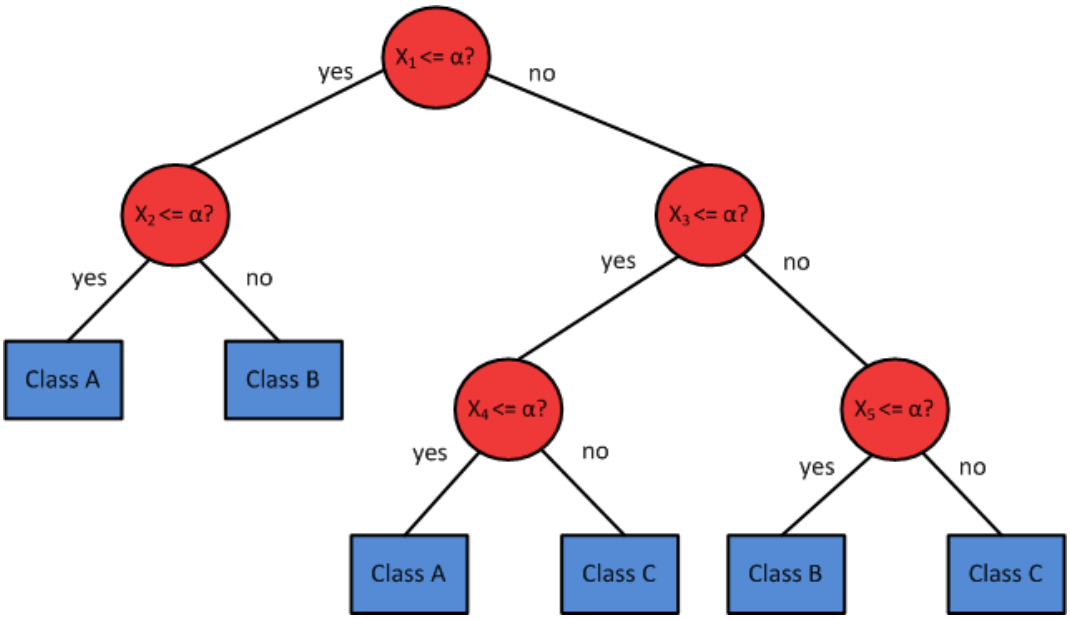


Figure 3.10: Illustration of the structure of a decision tree.

There exist different types of these decision trees. One of the most often used types is the so-called ordinary binary classification trees (OBCTs), which apply a sequence of decisions to individual features. These decisions are normally based on a question and a certain threshold value for the answer.

For example, such a decision might be based on the question is "$is feature x_i \leq threshold \alpha$_?". In a geometric interpretation, these decisions create hyperrectangles with sides that are parallel to the axes. Of course, other more complex rules are also possible that would then create more complex geometrical structures to split the feature space.

As mentioned before, these systems are based on a sequence of decisions. This raises the question, which decision should be made first. Or asked in a different way, if there is an optimal order of the decisions.

Regarding [[33]] the following four design elements are essential for building a decision tree in the training phase of a learning problem:

- At each node, the set of questions to be asked has to be decided. Each question corresponds to a specific binary split into two *descendant nodes.*
- *A split criterion must be adopted according to which the best split from the set of candidate ones is chosen.*
- *A stop-split rule is required, to control the growth of the tree, and a node is declared as a terminal one (leaf).*
- *A rule to assign each leaf to a specific class is required.*

In the following each of these essential elements of decision tree learning will be described in more detail.

The set of questions

Considering the OBCT type of decision trees, then the questions that have to be asked in each of the nodes is of the form "$Is feature\_\ x_i \geq threshold\_\alpha$". Theoretically, the sequence of questions that can be asked can be infinite since in each question the threshold values α could be chosen in an interval $Y_\alpha \subseteq R$. In practice, it is of course neither possible to ask an infinite number of questions nor would it make sense to ask extremely long sequences questions with extremely small changes in the threshold values.

Since the number of training features is finite, the set of possible questions is finite in consequence, too. From this set of possible questions the one for the best split has to be chosen. This split has to be done with respect to the splitting criterion.

Splitting Criterion

Every binary decision in a node splits nodes feature space into two distinct descendant feature spaces. According to the "YES" or "NO" answer of the questions in the node "t" the descendant nodes are denoted as t_Y and t_N in the following part. The node t is referred to as the ancestor node. Extending the tree by splitting the feature space from the root node down only makes sense, if every split generates a subset that is more "class homogeneous" compared to the ancestor node. In order to do this properly, a measurement that quantifies the node impurity and split the node so that the overall impurity of the descendant nodes is optimally decreased with respect to the ancestor node's impurity.

Such a measurement can be defined based on Shannon's Information Theory. For example regarding [[34]], let $P(w_i|t)$ denote the probability that a vector in the subset X_t, associated with a node t, belongs to class $\omega_i, i = 1, 2, ., M$. A commonly used definition of node impurity, denoted as I(t), is defined as:

$$I(t) = -\sum_{i=1}^{M} P\left(w_i|t\right) log_2 P\left(w_i|t\right) \tag{3.45}$$

It is easy to recognize that $I\left(t\right)$ takes its maximum value in case all probabilities are equal to $\frac{1}{M}$ and becomes zero in case all sample data belong to the same class. In practice, probabilities are estimated by the respective percentages, N_t^i/N_t, where N_t^i is the number of points in X_t that belong to class ω_i. The decrease in impurity is defined as [[34]]:

$$\Delta I\left(t\right) = I\left(t\right) - \frac{N_{tY}}{N_t} I(tY) \times \frac{N_{tN}}{N_t} \times I(tN) \tag{3.46}$$

Where $I\left(t_Y\right), I(t_N)$ are the impurities of the t_Y and t_N nodes. Recalling that the main goal is to find the best question from a set of questions, then the best question to choose is the one that performs a split that leads to the highest decrease of impurity.

Stop-Splitting Rule

As described before, it is also important to define a rule when to stop the splitting of a node and to therefore declaring it as a leaf of the tree. One possible way to make this decision might be to make this decision based on the previously defined $\Delta I(t)$. It is simple to define a threshold value T and stop splitting, in case the value of $\Delta I(t)$, over all possible splits, does not exceed T anymore. According to [[34]]: "Other alternatives are to stop splitting either if the cardinality of the subset X_t is pure, in the sense that all points in it belong to a single class."

Class assignmnet rule
Once a node has been declared to be a leaf by applying the stop-splitting rule, it then still needs to be given a class label. According to [[34]], a commonly used rule is the majority rule, that is, the leaf is labeled as where

$$j = arg_i^{max} P(\omega_i|t) \tag{3.47}$$

This basically means, that a leaf t is assigned to that class which the majority of the vectors in X_t belongs to.

In the following abstract sketch of a basic decision tree algorithm, all the steps described before are combined together:

1. Start with root node ($X_t = X$).
2. For each new node t.
    - For each feature $x_k, k = 1, 2, ..., l$.
        - For each threshold value $\alpha_{kn}, n = 1, 2, ..., N_{tk}$.
            - Generate X_{tY} and X_{tN} according to the answer in the question: is $x_k(i) \leq \alpha_{kn}, i = 1, 2, ..., N_t$.
            - Compute the impurity decrease ($\Delta I(t)$).
        - End.
        - Choose α_{kn_0} leading to the maximum with respect to x_k.
    - End
    - Choose x_{k_0} and associated $\alpha_{k_0 n_0}$ leading to the overall maximum decrease of impurity.
    - If the stop-splitting rule is met, declare node t as a leaf and designate it with a class label.
    - If not, generate two descendant nodes t_y and t_N with associated subsets X_{tY} and X_{tN}, depending on the answer to the question: is $X_{k_0} \leq \alpha_{k_0} \leq \alpha_{k_0 n_0}$.
3. End

This concept of a decision tree is the foundation for several decision tree learning algorithms such as for example ID3 and its successor C4.5. Since these two algorithms are used very often in practice a short sketch of these two algorithms will be given in the following.

The ID3 algorithm [[29]] is based on the entropy and information gain measurement. Its name "Iterative Dichotomizer 3" is retrieved from its iterative class separation behaviour.

This algorithm is based on three basic steps:

1. Determine the Entropy for all unused attributes with respect to the training samples.
2. Choose the attribute with lowest entropy (or equivalently with highest information gain).
3. Create tree node based on this attribute.

As mentioned before this algorithm is based on two metrics, namely the Entropy and the Information Gain.

The Entropy is computed/denoted as follows:

$$E(S) = -\sum_{j=1}^{n} f_s(j) \, log_2 f_s(j) \tag{3.48}$$

In this equation S represents the set of samples. Variable n is the number of different values of a certain attribute in S and $f_s(j)$ describes the portion of the value j in the set S.
In case a set is perfectly classified then it has an entropy of 0.

The Gain is denoted as follows:

$$G(S, A) = E(S) - \sum_{i=1}^{m} f_s(A_i) \, E(S_{A_i}) \tag{3.49}$$

In the above equation S again is the set of samples.
$G(S, A)$ is the gain of the set S after a split over a certain attribute of A.
$E(S)$ is the previously described information entropy of the set S.
Variable m is the number of different values of the attribute A in S.
S_{A_i} is a subset of S containing all items where the value of A is A_i.
Function $f_s(A_i)$ is the portion of items having A_i as value for A in S.
A_i is the i^{th} possible value of A.

The C4.5 algorithm, developed by J. Quinlan [[30]], is a successor of the ID3 algorithm described before. This algorithm builds decision trees from a set of training data in the same way as it is done with ID3 based on the information entropy metric. The main difference to the ID3 algorithm is, that the information entropy is used in a normalized form as a splitting criterion.

The following pseudocode of C4.5 describes the working principle of C4.5 in more detail:

1. Checking the base cases:
   - If all samples belong to the same class, create a leaf node for the decision tree.
   - If all nodes provide zero information gain, create a decision node higher up in the tree by using the expected value of the class.
   - If an Instance of previously-unseen class encountered, create a decision node higher up in the tree using the expected value.
2. Determine the normalized information gain for each attribute from splitting on it.
3. Choose the attribute with highest normalized information gain and create decision node that splits on this node.
4. Recourse on the sub lists obtained by splitting and add these nodes as child nodes.

Independent on which decision tree algorithm is chosen in practice, all have some advantages and disadvantages in common. Advantages of decision trees is that in general they are human readable. That means that they reveal the internal structure of the learned model. The understandability might be limited due to the fact that these models learned can be represented in very large decision trees. A major disadvantage of decision trees in general is their high variance. This simply means that small changes in the data set will cause a completely different tree to be learned.

As described before, the size of such a tree is one of its crucial factors. In order to cope with this problem a technique called pruning has been developed, which can be used to reduce a decision trees size. The strategy behind this technique is to let a tree grow large and then prune, based on pruning criteria. Commonly, the criterion is based on combination of error probability estiamtion with complexity measuring[[33]]. Since this technique is not actively used within this thesis it will not be described here.

Another very useful concept regarding decision tree learning is called (Random-)Forests.

Random Forest uses the idea of bagging in tandem with random feature selection. This means that different decision trees are learned in parallel, but each just on a subset of the provided features. Because of this reduction in the number of used features, the depth of the trees is smaller and the models are simpler. This also means that the model of a single tree is less correct but more general. It is important to mention, that each of the trees is normally fully grown, that means no pruning is done, as with other decision trees. In addition, it is also important to notice that a random subset for training is selected for each tree. This selection is done with replacement. In the last step, the models of the trees are combined.

Due to this principle, this model construction and learning approach is known to have very good generalization capabilities. This basically means that it is more robust against changes in the data set. Besides this, it basically has all advantages that other decision trees also have. Some of these advantages are for example the easy implementation, its classification speed and its easy configuration.

3.3.5 Naive Bayes Classification

The so-called "Naive Bayes classifier" is a probabilistic classification concept that is based on Bayes'theorem with a very strong independence assumption on the features. This basically means that it assumes that no correlation exists between different features of a feature vector and that learned model for classification is based on conditional probability. This means, assuming two different events A and B, the model that is learned by this classifier is the conditional probability that one event (e.g. A) occurs under the condition that A already occurred.

Regarding [[24]], this concept can be extended by having a variable that is dependent on numerous other variables. In general this leads to a set of equations:

$$p\left(A \cap B_1\right) = p\left(A|B_1\right) \cdot p(B_1) \tag{3.50}$$

$$p\left(A \cap B_2\right) = p\left(A|B_2\right) \cdot p(B_2) \tag{3.51}$$

$$... \tag{3.52}$$

$$p\left(A \cap B_n\right) = p\left(A|B_n\right) \cdot p(B_n) \tag{3.53}$$

This set of formulas could also be denoted as one combined formula in the following way:

$$\sum_{i=1}^{n} p\left(A \cap B_i\right) = \sum_{i=1}^{n} p\left(A|B_i\right) \cdot p\left(B_i\right) \tag{3.54}$$

Assuming a finite set of events for B_i the equation can be transformed to:

$$\sum_{i=1}^{n} p\left(A \cap B_i\right) = p\left(A\right) \tag{3.55}$$

This equation can then be reduced to:

$$p\left(A\right) = \sum_{i=1}^{n} p\left(A|B_i\right) \cdot p\left(B_i\right) \tag{3.56}$$

Considering a case where the occurrence of A depends on only two mutually exclusive events, which are B or $\neg B$ (logical not B), the probability of A can be described in the following way:

$$p\left(A\right) = p\left(A|B\right) \cdot p\left(B\right) + p\left(A|\neg B\right) \cdot p(\neg B) \tag{3.57}$$

The probability of event B could be described in a similar way:

$$p\left(B\right) = p\left(B|A\right) \cdot p\left(A\right) + p\left(B|\neg A\right) \cdot p\left(\neg A\right) \tag{3.58}$$

Now this term could be substituted into the Bayesian rule:

$$p\left(A|B\right) = \frac{p\left(A|B\right) \cdot p\left(A\right)}{p\left(B|A\right) \cdot p\left(A\right) + p\left(B|\neg A\right) \cdot p\left(\neg A\right)} \tag{3.59}$$

The Naive bayes classificator

The Bayesian rule explained previously can also be useful when it comes to data classification tasks and is indeed used in classification systems such as SPAM filters. In general, Bayesian theory provides the possibility to calculate a degree of belief. A Bayesian classificator expresses its degree of belief in certain proposition of interest by using a single real number.

Later within this thesis a classifier called na_ive Bayes will be used. This is a simple probabilistic classifier which is based on the before described Bayes Theorem. It is called na_ive because it is based on a very strong(naive) independence assumption of the variables.

This simply means that the naive bayes classificator assumes that the presence or absence of any feature within a feature vector is independent of the presence or absence of any other

feature within this feature vector. This assumption is somehow naive, because it is unlikely to hold true in many cases in reality. Hence it extremely reduces the complexity of a classification problem.

In naive bayes classification, each data sample is represented by an n-dimensional vector of features. The classification task is to assign each of these feature vectors to the class $c_i \in C$, to which it belongs with the highest priority.

Formally this is denoted as:

$$p\left(x|y=c\right) = \prod_{i=1}^{D} p\left(x_i|y=c\right) \tag{3.60}$$

This formula expresses that the na_ive Bayes classifier is simply based on the product of the probabilities of all features of a feature vector to calculate the overall probability for a given feature vector that it belongs to a certain class (label).

This classification approach has some advantages as well as some serious drawbacks. The advantage is quite obvious. Due to its simplification of the classification problem it is very easy and fast to compute the model and applying it on an actual data set. Because of the independency assumption of each of the features it is applicable on very high dimensional data sets. It is even possible to apply this classifier on data sets with a few thousand attributes/features. Furthermore, the probabilistic model can be continuously updated in case new labeled data is available at a later time.

This simplification also introduces some serious drawbacks. The main disadvantage of this classifier is that its simple assumption is not applicable in many real world scenarios, because many data sets contain some dependency between their features. Therefore, this classifier is very likely to suffer from a problem that is called "oversimplification". This simply means that the classifier builds a model of a classification problem that is far to simple to perform well on real data sets. Furthermore, since this na_ive Bayes classifiers are based on probabilistic traits of a data set it is very likely to fail in case it is applied on imbalanced training data sets.

This means that the data set contains much more samples of one of the classes then of the other one. In such a case of imbalanced training data the classifier would learn the imbalance as part of the model. Assuming the classifier is applied on a data set with two classes (A and B) and class A consists of 10 times more samples, then the classifier would learn that class A is 10 times more likely to appear then class B. Hence, on the later actual data set this distribution might not hold true any more. This behaviour becomes obvious when recalling that the classifier determines in general which class appears how often related to the other class. Furthermore, it computes the probability that the data sample belongs to this class. Both probabilities are combined by Bayes theorem. Bearing this in mind, then it becomes very clear how important a balance training data set is for this classifier.

3.3.6 Ensemble Learning and Decision Fusion

Previously in this chapter, several distinct classification methods such as support vector machine or multilayer perceptron were presented. Each of these classifiers has certain advantages and limitations. In practice, one would like to combine the advantages of different classification approaches. Because of this, different methods were developed to combine multiple classifiers

into one classification system. This is also known as ensemble learning and decision fusion.

In practice, a used approach is to run different classification algorithms in parallel on the same data set and to combine their decisions by letting voting. Such a vote can be for example a simple majority vote or be based on different weights for each classifier. These weights for the current classification sample are usually determined based on the detection accuracy on previous classification samples. A further commonly used concept is based on determining the arithmetic average of all decisions.

Another very famous ensemble learning principle is called "Boosting". This method combines multiple weak classifiers in a way so that the new error of the new classifier is reduced to a minimum. Each of the weak classifiers only relies on one variable or attribute. Because of this principle, the model construction can be done very fast.

In addition to this, also a method called "Bagging" (Bootstrap aggregating) exists. Bagging is based on the principle to combine different regression and classification models into one system. Each of these models normally is considered based on a weight. That means that good models are included into the general model with a much higher probability. This principle is for example used in in the previously mentioned "Random Forest" algorithm.

Besides this also "Stacking" is a well-known concept for ensemble learning. The stacking concept is based on the sequential concatenation of different machine learning algorithms. This means that a first classifier, such as a SVM, is applied to some input that has to be classified. In the next step the classification output of this classification algorithm is used as classification input to another machine learning algorithms, such as a Bayes classifier or a perceptron.

After this introduction of the fundamentals of machine learning, the next section will provide the necessary fundamentals for simulating network traffic that will be used later in this thesis.

3.4 Simulation

In the previous section - about theory of classifying data - the importance of reliable training data was mentioned. To provide such a reliable and controllable set of training data it is essential to set up a simulation environment.

Such an environment has to meet at least the following four requirements: it must be flexible, controllable and well understood (by means of simplicity) and close to reality as well. This section describes the meaning of these terms in the context of NAT detection in more detail before an overview of possible different topologies is presented.

3.4.1 Preliminaries

There are numerous requirements a simulation environment has to fulfill, especially when it comes to such a complex domain such as producing network traffic that is close to reality. This task is very complex because there is no standard user behaviour that could be simulated. Instead, the traffic varies a lot between different users. Even if assuming that traffic of similar users must be similar, this task is not trivial at all. For example, when trying to distinguish between private and business users each of these classes still stays very complex. But not only is there a huge variance in the behaviour of the users themselves, but also the used hardware

varies a lot. For example, the hardware setup is never the same when considering different enterprise networks, because each enterprise has its individual requirements that might depend on the companies' size and the business field they operate.

Therefore, it is important that a simulation environment is flexible enough to be easily reconfigured to meet different user groups or hardware setups. As described before, besides the flexibility, there are also other important requirements a simulation environment has to meet. Just to mention the most important ones, it should be close to reality and it should be fully controllable and understood. Figure 3.11 illustrates these requirements.

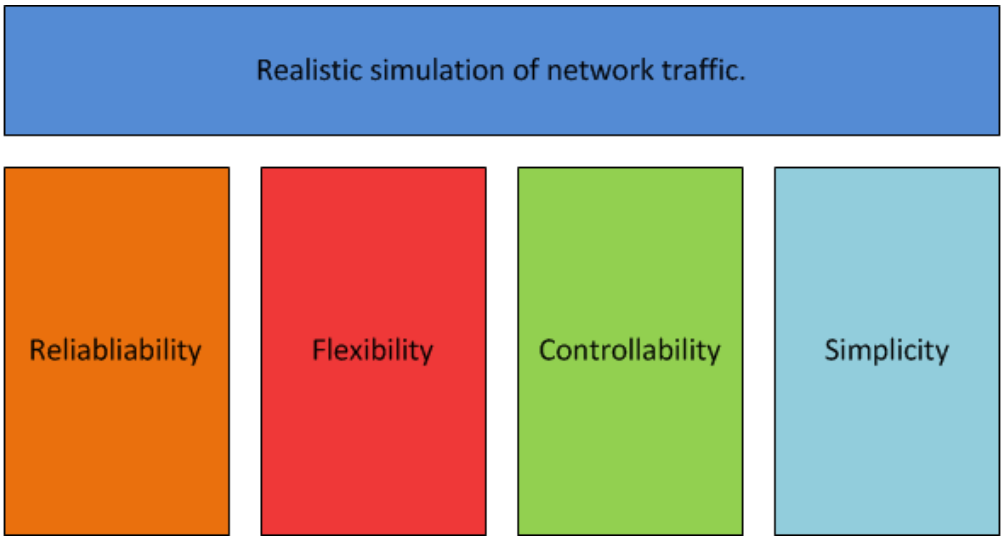


Figure 3.11: The four fundamental requirements for a reliable simulation.

Flexibility in this context means that it should be easy to adapt the simulation environment depending on the needed data. In the context of NAT detection in general this means for example that the NAT-gateway has to be replaceable by a different one. This is important to be able to make behavioural differences between different NAT-Gateways visible and understandable under exactly the same conditions. Hence, flexibility in this domain is not only related to the NAT-gateway itself, but also to the number of connected hosts, the used software as well as the accessed services.

Close to reality means that the simulated user behaviour and hardware are the same or at least very similar to the hardware used in reality. The same holds for the user behaviour and the accessed services. This is because the behaviour of the user(s) should be comparable to real users and the software used should be commonly used in reality too. It is very important because, as described before, the main goal of the simulation is to provide well understood and labeled training data for the classifiers. Furthermore, it is obvious that any biased or too artificial data will cause the classifiers to perform badly or completely fail on real data.

Fully understood in this context means that the presence of each part of the simulation environment and its influence on the resulting data is known and explainable. Therefore, this trait can also be called simplicity. This is the condition where the simulation completely differs from traffic that is captured from a real ISPs network. Because neither is there information about how many users share one IP nor is their behaviour known or even controllable and normally also the used hardware is not known.

Having a controlled environment in this context means that unintended packets from the outside (Internet access point and behind) are blocked and do not reach the simulation environment and therefore also do not spoil the data set. Such unintended traffic could be normal network management traffic as well as any abnormal or malicious traffic send from the outside, such as for example a port scan.

After having seen the fundamental requirements of a simulation environment in this section, the following section describes a basic simulation topology that can then later be adapted for setting up simulation environments.

3.4.2 Setup of Simulation-Environment / Topology

After pointing out the essential requirements a simulation environment has to meet, this section now provides examples of different possible setups and discusses their usefulness with respect to these requirements.

First of all, it is important to distinguish between different "degrees" of simulation with respect to the several components of the setup. In general every simulation environment consists of at least 4 parts, which are the user(behaviour), the host(hardware and operating system), the NAT-Gateway and the remote station/Internet. Figure [3.12] illustrates this principle.

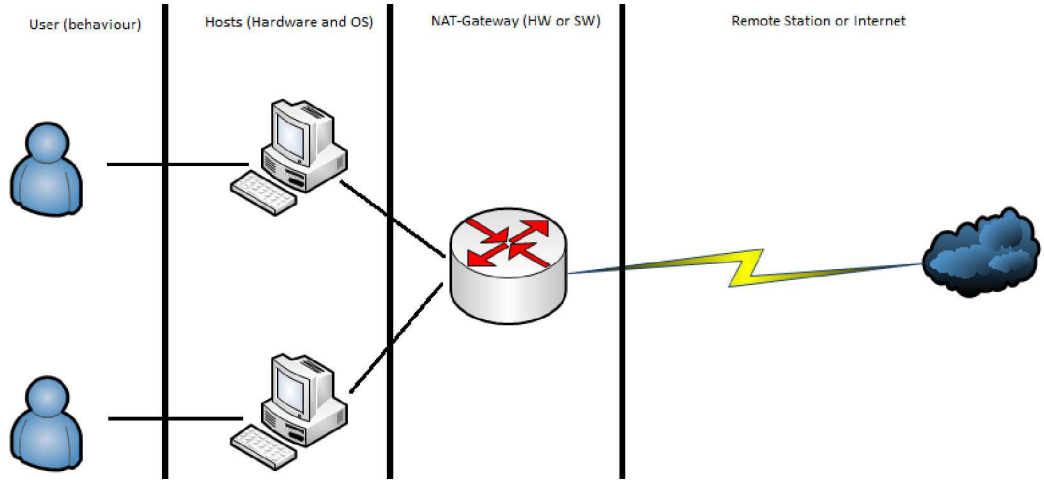


Figure 3.12: Illustration of the four fundamental simulation parts.

When further talking about simulation, it is essential to keep in mind that each of these parts can be simulated or emulated respectively in different ways. Having different options to simulate each of these components leads to a large number of possible different setups in theory. Hence, in practice not all of them are realistic or they simply differ enough from the others.

Starting from the left in the figure [3.12] the user (behaviour) can be created in different degrees of simulation, starting from fully artificial changing to partially artificial, replaying captured real "user(behaviour)" and ending in the possibility to let real users create real traffic in real time - but still in a controlled environment.

The next pillar of this fundamental topology is the host (hardware and operating system) which executes the desired actions from the user(s). As before, this part of the simulation environment can be a real hardware machine running a certain operating system and other user applications. Such a setup would be completely real, but of course there is also the possibility to emulate the hardware by using a virtualization software such as VirtualBox [[45]],

VMWare[[50]] or Parallels[[26]]. In the description above the OS is also part of this component, hence this part is not seen as virtualizable but exchangeable part of a host.

The NAT-gateway is the essential part of each possible simulation setup, but also the most problematic in the way that there are many manufacturers providing NAT solutions either in software and hardware gateways adopted to different user groups and use cases. Therefore, it is clear that the number of gateways that will and can be part of such a simulation is limited and could not cover all possible specialties that are present in real world networks. Hence, even this component could be classified into three major classes. It could be either a hardware NAT-Gateway or a software-gateway or the software of a hardware NAT-gateway running on emulated hardware.

On the rightmost side in the topology presented in figure [3.12] there is still the remote side left. This is the part which the host(s) from the left most side will communicate with. This part can either be a machine accepting connections and/or providing certain services or it could be connected to the Internet and therefore allowing the hosts to access virtually any service or resource on the Internet.

Some of the solutions described above are fully simulated, some are hybrid and some are solutions completely based on real hardware. Figure [3.12] provides an overview of all the abovementioned categories. But each of these setups meets some of the above mentioned requirements better than the others do.

For example, on the one hand the completely simulated/emulated versions perfectly meet the requirement to be very flexible and well understood, because each component can be replaced by another to change the setup or verify the results. On the other hand choosing a setup completely based on real hardware and real users producing the traffic would have the advantage to be very realistic but less controllable and it would not be possible to reproduce the behaviour in exactly the same way.

The aim of this section was to present the main concepts and requirements for a simulation setup. The actual configurations used within this thesis are described in Chapter 4 "Extraction of Sample Data".

3.5 Overview of Principal Components

After the description of the different methodologies and components in the previous sections of this chapter, this section now describesthe overall overview of how all these components work together and how the data and decision flow looks like in general.

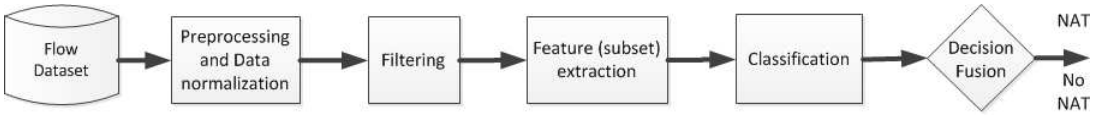


Figure 3.13: Principal Components and Data Flow

Figure [3.13] illustrates these distinct parts of a general NAT detection system.

As mentioned before in this chapter, there are different ways of generating/extracting the Flow data, but for the general system description it is assumed that the sampling, exportation and collection is already done. It is important to keep in mind that the actual detection process

starts with preprocessing the flow data. This preprocessing can consist of different parts such as adapting the date and time format or also the IP address representation. This is important since some flow exporters tend to transfer some of the IP addresses in the standard format of four octets separated by dots (xxx.xxx.xxx.xxx) and sometimes the same flow exporters provide IP addresses without these dots. This makes it necessary to normalize all IP addresses to one of these formats.

Assuming that the data set now is in a normalized format the next step is "Filtering", which is essential when it comes to performance considerations of the detection system. This component filters out for example the traffic for a specific IP address from large data sets. Of course, there could be several filtering criteria's such as only outgoing flows of one special IP address and only flows related to a specific protocol. It is obvious that the filtering has a huge effect on the amount of data that has to be processed in the later components, which then affects the performance of the overall system.

3.6 Summary

In summary, this chapter provided theoretical background knowledge on various machine learning aspect, such as the feature creation process and different machine learning algorithms.

First Section 3.1 described the feature selection process in detail. In this section different steps in the feature selection process were presented.

Then in Section 3.2 a short overview of potential ways how features could be introduced, to the data sets used later in this thesis, were presented.

Section 3.3 then provided a detailed description of the machine learning algorithms used in later parts of this thesis.

In addition to this section 3.4 provided an overview of the fundamental requirements a simualation environment has to fulfill in the domain of network intrusion detection and NAT detection.

As the last part of this chapter, in section 3.5 the principal components that are necessary for the NAT classification process were identified and grouped.

CHAPTER 4

Extraction of sample datasets

This chapter describes the methodology that was used to produce suitable sample data sets to recognize the effects introduced by a NAT device. In the following sections, two different approaches to extract sample NetFlow data are described in more detail.

4.1 Preliminaries

Extracting sample NetFlow data is an essential first step towards a fully automatic detection approach. But, as described before, it is important to keep in mind that these sample data sets have to meet several requirements.

Suitable in this context means that the extracted data samples must meet several requirements. First of all, the extracted data samples should of course be close to reality. Secondly, the data samples should also come from a controlled/adjustable and well understood environment.

However, it will be difficult or almost impossible to fully meet all these requirements at the same time. Hence, the extraction of sample data will in general be done by following two different approaches. Each of these approaches will completely meet at least one of the requirements described before, but of course at the same time it will meet the remaining requirements only partially or not at all.

The first approach was a simulated environment, which perfectly meets the requirement Adjustability. Of course, at the same time it is also well understood, because it is known, if the extracted data was modified by a NAT device or not. Furthermore, other details are also known very well, such as the number of hosts that were connected to the NAT device, their operating systems and the used software to produce the network traffic. However, this approach has the drawback that, no matter how much it is adapted, it will stay more or less artificial.

The second approach was to sample and export NetFlow Samples from a German Internet service provider. This approach of course has the huge advantage that the data samples come from real Internet users and therefore perfectly meet the requirement to extract samples that are close to reality. However, these data samples come with the big drawback, that the environment they are extracted from is more or less unknown. This means that the data samples are neither well understood nor adjustable.

After this overview of the two different approaches of extracting NetFlow samples and the requirements they have to meet, the following sections of this chapter describe the two approaches mentioned above in more detail.

4.2 Simulation

As described before, the simulation approach was chosen, because it provides a very well understood and adjustable environment to extract NetFlow samples from.

Practically, "well understood" in this context does not only mean the knowledge of the used components, such as the number of hosts and their software setup, that created the network traffic, but also that it is possible to capture the traffic that comes from each component at a time in parallel. Doing this provides the possibility to exactly determine how each component - and especially the NAT gateway - influences the final result.

It is important to keep in mind that there are different ways to set up such an environment and depending on the approach that was chosen there might be differences in how adjustable and how well understood the particular setup really is.

Even though until now only extracting samples from a simulated environment and from an Internet service provider were distinguished in general, this section will now describe three different approaches that were set up and used to extract NetFlow samples within this Thesis.

Each of these setups was selected with respect to the requirements described above and the simulated NAT-Gateways are commonly used in practice at the time when this thesis was written (2012/13). Each of these approaches pay a bit more attention to a specific use case or user group.

The first setup aims to simulate NAT-gateways that are likely to be used in a business environment. Afterwards, the second approach aims to simulate a common setup for private residential NAT-gateways. The third approach is based on desktop virtualization software that allows the user to run a virtual machine inside his host machine and share the same IP address to access the Internet. This last approach is used in both private and business environments today.

4.2.1 Simulation Methodology

In order to meet the fundamental requirements of a simulation, that were presented before it is necessary to carry out the simulation based on a suitable methodology. One very important requirement to the simulation setup was to have reproducible results and in addition to have a flexible and well understood setup. Furthermore, of course, the results of different runs of a simulation or changed setups should still be comparable.

To reach these goals as precisely as possible, all simulations were based more or less on the basic topology presented in the fundamentals. Only slight changes were made from one run of the simulation to another. This means that for example the number of hosts was only increased or decreased slightly between different runs. This allowed to discover effects that were introduced by each component. Furthermore, the user behaviour was created in a very similar way on each host. This means all hosts and operating systems that were used were more or less in their standard configuration. Each operating system was allowed to download and install updates automatically and in addition on the used windows host Avira Antivir [[2]] was installed and allowed to update itself.

In addition to this, the hosts were running an E-mail client software and further user related traffic such as browsing was either generated based on a browser test frame work or by hand but in a similar way. This means that for example the same set of Web resources was accessed on each host, even though the browsing was performed by hand by a real user.

During a simulation the traffic was captured on various points of the setup. For example, the traffic of each host was captured by using either tcpdump [[43]] on linux or Wireshark [[49]] on Windows. These packet captures were then transformed into flows by using the Linux software softflowd [[39]]. In addition to this, the emulated Cisco router was configured to sample and export flows as well. These flows were captured by a flow collector software classed nfdump (nfcapd in particular).

This procedure allowed to exactly compare the traffic that was produced by the hosts with the traffic that was then put out by the NAT-Gateway. For such a comparison further tools for visualizing the data were used. Sub sequentially, the flows were passed to the classifiers that are presented later in this thesis.

In the end, applying this methodology ensured that the requirements presented before were almost completely met. For example, the capturing of the network packets before and after the NAT-Gateway allowed to understand the effects that were introduced by the NAT-gateway itselfe.

4.2.2 Business equipment

The aim of this approach was to simulate and analyze the behaviour of NAT-gateways typically used in business environments. It is important to keep in mind that enterprises that use this kind of equipment may own and use a pool of IP addresses for accessing the Internet. The simulation environment described in this thesis only makes use of a single outside interface and IP address to have comparable results with the other two simulation approaches even though the operating software of the simulated routers would support this feature.

This approach is based on the emulation software "Dynamips" [[10]] that is capable of emulating Cisco hardware and running Cisco IOS images on top of it. The hosts connected to this "NAT enabled Router" are simulated by an adaptable number of Virtual Machines emulated with VirtualBox [[45]]. Both of these emulation software's are controlled by the Software called GNS3 [[15]]. GNS3 interconnects both (the virtual machines and the emulated Cisco Router) through tunnel interfaces.

The figure 4.1 illustrates the described topology of the involved components.

For the sake of reproducibility of the results, a test framework was created based on several simulation and test components.

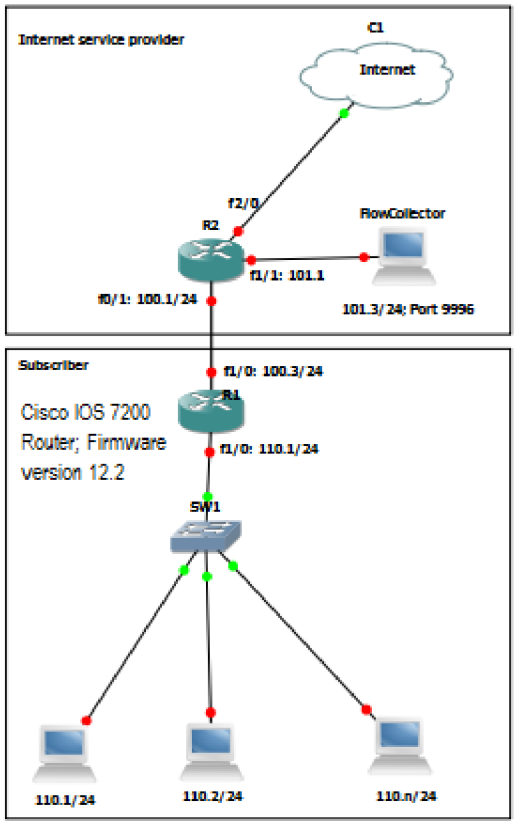


Figure 4.1: Topology of business equipment simulation.

The most important component of the fully simulated environment was the emulator component for Cisco IOS Images, which was based on Dynamips [[10]]. This software is able to emulate different Cisco Hardware components such as several routers and hardware interface slots of these devices. Furthermore this software is able to execute several types of original Cisco firmware images on top of this hardware emulation. In addition, GNS3 [[15]] was used as a graphical frontend to the emulation software. Hence GNS3 not only provided a graphical frontend to the emulation part of the simulation environment, but also integrated the Virtual Box software, which emulated the hosts was creating the actual connections.

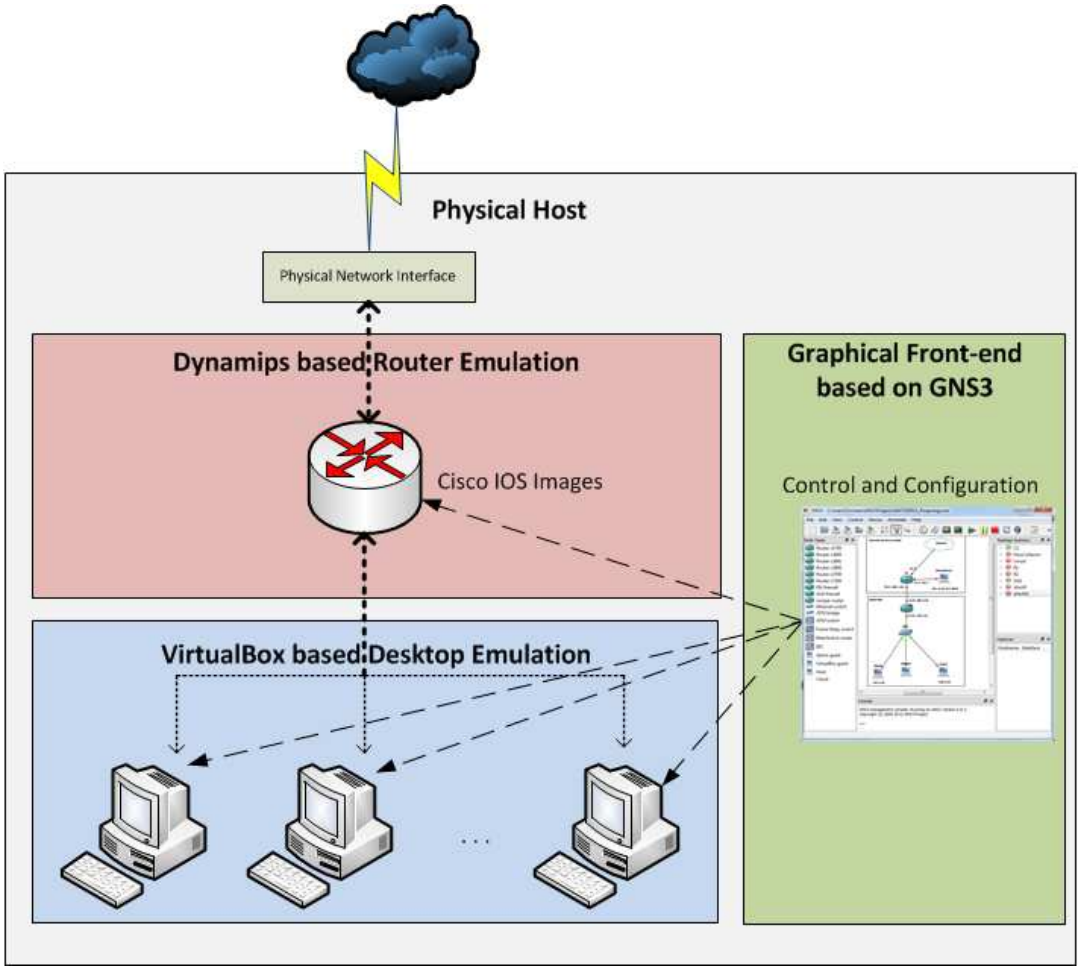


Figure 4.2: Overview of simulation components and interaction.

Figure 4.2 illustrates the previously described compontents and their interaction with each other. The advantage of this setup is that it can easily be adapted in terms of adding hosts or changing the router version that is used for the simulation. For the simulation of the Cisco router an IOS 7200 Image of version 12.2 was used. The number of hosts and the used operating systems were changed in different runs of the simulation.

Furthermore, it is also important that this simulation environment had connection to the Internet in most setups. Only in a few special setups the connection to the Internet was replaced by a host/server that accepted special connection requests such as for example telnet connections. This shows another important aspect of the simulation environment, due to its flexible changeability, it was possible to analyze how certain applications influenced the later

retrieved flow data set. This of course is very helpful, especially for finding patterns that indicate the presence of a NAT-gateway. This is even more supported by the fact that it is possible to capture the traffic at different points in the topology and then convert it into NetFlow later, for analyzing the changes that were introduced by each component on the exactly the same communication/connection(s).

Different appllications were used to create the actual traffic of the hosts. More details about this part of the simulation are described in the section "Generating Traffic" later in this chapter, because the technique used was similar in different simulation setups.

4.2.3 Private Residential Equipment

This approach aims to simulate a NAT setup commonly used in private usage. In order to make this approach as realistic as possible, a real hardware router that is commonly used in this kind of local networks was used as a NAT-gateway. This was done because there are no emulators for this kind of routers publicly available.

However, just using one of these routers is not enough, because these devices are normally not able to sample and export NetFlow directly. Thus this router is not directly connected to the Internet but through a network bridge.

This network bridge was set up on a Ubuntu host with two Ethernet interfaces. In particular, adding this component to the simulation environment makes it possible to sample and export NetFlow directly on this Ubuntu machine. The sampling of the NetFlow data was done by the freeware "softlowd" [[39]] which used the local host as the target flow collector. Collecting the flows was done by the "nfdump" package (nfdump and nfcapd).

4.2.4 Generating Traffic

After describing different simulation setups by means of their network setup, the focus of this part will be on generating the network traffic which is then sampled to flows in each of these different setups.

Of course, it is possible to create the traffic by hand for a limited number of hosts. This approach works fine of course for a single host but may already get hard if a second host is introduced to the setup - depending on the applications/traffic that will be used in the simulation.

On account of this, it is very helpful to automate the whole task of creating network traffic or at least parts of it. Furthermore, automating this task has the advantage that the exactly the same behaviour/sequence could be (re-)produced in each of the simulation setups in the exactly the same way.

This deterministic approach may seem very artificial at a first glance and indeed it is, but it is very helpful with produce comparable results in all the setups. Since the main advantage of the simulated environment is that things can easily be changed, it is of course also possible to extend the automatic network traffic generation by introducing random behaviour in a later step, which then comes closer to the real user behaviour.

For creating traffic different approaches were combined. One part of this approach was regular polling of E-mails through an E-mail client software. For each setup the open source tool Thunderbird [[22]] was used because besides the fact that it is open source (and therefore allows to analyze its behaviour by dissecting its source code, if necessary) it is available for different operating systems. Another important reason for choosing this software is of course

also that it is commonly used today.

The same argumentation as for Thunderbird also applies to the selection of Firefox as Web browser.

However, in contrast to automatically polling E-mails by using an E-mail client, automatically browsing the Web with a browser is normally not natively supported by browser(s). Therefore, a framework called "Selenium" [[35]] originally designed for automatic GUI based Browser testing was used to provide the capability of automated browsing to the simulation environment. It turned out to be useful to simulate user behaviour for the sake of creating traffic in a way that is very similar to the way it would be done by a real user. In the end the frame work was originally designed to control a browser in the same way a user would do it. This framework provides several options for this task. One very easy approach to use this framework is to record a browsing session of a real user and then replay it. A more advanced option is to control the framework by a programming or scripting language such as for example perl or python.

This approach provides more control over the browsing and it is for example possible to add more variance in the user behaviour then exactly reproducing the same session every time. Hence it still remains to be some sort of artificial, because one has to know the Web site input fields in advance to for example stay and click on a Web site. This is because the framework is normally used for testing one particular Web page during development time, which means that each element of the Web site is well known. However, if the intended use is to automate a realistic browsing session over some minutes or even hours, this becomes a rather complex task, since the fields to access on each Web page would have to be known in advance of the simulation.

It is important to find a good balance between simulating realistic browsing behaviour and still keeping the implementation effort at a reasonable level. For this reason, the user behaviour part of the simulation was done in the following way. A number of short browsing sessions was recorded from a real user at a first step. In a second step, a small script was developed that chosen randomly from these sessions. The advantage of this approach was that the implementation was kept to a minimum level, but the simulation could be running for an arbitrary length and was guaranteed to be relatively realistic since it was composed from real user behaviour.

In addition, automatic outgoing E-mail traffic was produced by a python script that uses the python module "SendMail". This tool was chosen because the same script using the same module can be used on different operating systems. Of course, other programming/scripting-languages could provide the same capabilities.

Furthermore, the automatic update mechanisms of different operating systems create traffic which is not always completely controllable. Moreover, also Antivirus Software was installed on the machines, because this software also regularly establishes connections to check for updates.

4.3 Real ISP data

One of the goals of this thesis is to provide the possibility to estimate the size of botnets more precisely based on NetFlow data. Since the Botnet detection is normally done in networks of the Internet service providers, it is a good idea to apply the classifiers extracted from the simulated environment on NetFlow data from a real Internet Service Provider as well.

As described in the fundamentals about sample data extraction, each sample data set has to meet several requirements. The NetFlow samples from the ISP obviously completely meet the requirement to be close to reality because it is the reality. At the same time neither is this data set controlled nor a precise knowledge about the components being part of the network.

Furthermore when dealing with real data samples from an ISPs network then also processing the amount of data could be a problem and patterns clearly visible blured just because of the huge amount of data at a time. This is why in this domain more data does not necessarily mean more knowledge/information.

The data from the ISP was sampled in NetFlow version 5 for one business customer who used one static IP address for his business. The NetFlow data was present obviously sampled in slots of five minutes every ten minutes. In addition to the flow samples, the ARP tables were extracted. This provided some information about the internals of the network, such as the number of hosts that were present at a certain time. Based on this information source, it was possible to observe the maximum size of the network. Regarding the ARP tables the maximum number of hosts that were used in parallel was 44 hosts, which shared this one static IP address. The flow data was sampled over a period of ten days.

The main advantage of this data set is obviously that it perfectly meets the requirement of being realistic, since it is retrieved from a real network and represents the real usage of this network. Hence there is no detailed information about the single hosts that used the network, such as the used operating system, applications running or firewall software that might influence the network traffic on the hosts. The only further information that was available was that the customer used a Fortinet-firewall to connect the hosts to the internet.

4.4 Conclusion

This chapter provided some important information about the way sample data sets were retrieved within this thesis. It started with briefly discussing the different requirements a good sample set has to fulfill in this problem domain. After that, the extraction of sample data was described in more detail for two fundamentally different approaches. The first approach discussed in detail was the simulation of traffic and NetFlow respectively. This approach was further split by the two use cases - business equipment and private residential equipment. The other main approach described was based on retrieving sample data sets from a real network of a business customer of a German ISP.

It was pointed out that each of these approaches has its advantages, by means of fulfilling certain requirements better than others as well as some drawbacks. For example the simulation was very flexible, well understood and provided the possibility to see the influence of any change in any of its components. In contrast to this the ISP data was less flexible and did not provide the same possibilities, but it was closer to the reality, since it was based on real user traffic from a real network.

In the end, keeping these differences in mind, each of these setups reaches its particular goal.

CHAPTER 5

Analysis of NetFlow data sets

This chapter provides a detailed analysis of the data sets that were used in this thesis. At first, a theoretical description and discussion of the data sets is provided. Then an analysis of the simulated and the ISP data follows. Finally, the extracted feature set for all the classifiers is presented in detail.

5.1 Theoretical Description and Discussion of the Data Set

This section provides an overview of the data fields of the Cisco NetFlow Version 9 that are considered to be relevant for the proposed approach of detecting NAT. A full description of the data fields described in this Standard can be found in the Appendix. This overview also consists of a short discussion, outlining the reasons why each data field is considered to be useful for detecting NAT with the proposed approach.

In general, it is important to keep in mind that variables that should be used for detecting NAT should either be changed in the NAT-gateway itself, changed in different ways by an operating system, or should be influenced by the user behaviour. Data fields that are influenced by the NAT-gateway in a unique way are probably the most significant features, but since NetFlow is a highly aggregated data format these effects - if any - will not be recognizable easily. The same holds for data fields that could be influenced by different operating systems (OS) in a unique way. The idea with this kind of data fields is to recognize the presence of different OS at the same time, which is an indicator that there are several computers sharing the same IP address.

Another approach - as proposed earlier- could also be to detect NAT by detecting patterns in the user behavior that are unique for single users or respectively patterns for multiuser behavior. The basic assumption here is that more users should create more traffic. The traffic is assumed to be generated more equally over time. This approach then involves generating mod-

els for single user and multi user behavior. Of course, this can only be a fuzzy probabilistic way.

Now, all data fields that are considered to be relevant are listed with a short description also providing the reason why each of them is considered to be useful. For the following description, a client-server communication will be assumed, because this is the most often used type of communication today.

Affected by the NAT-gateway:

- **"sa"** - Source Address: The IP address where the traffic comes from. In case of outgoing traffic this address will always be the same. In case of incoming traffic it is the server's IP address. This is a fundamental field to describe the communicating parties and therefore essential information to describe a flow.
- **"da"** - Destination Address: The IP address where the traffic goes to. In case of incoming traffic this address will always be the same. In case of outgoing traffic it is the server's IP address. This is a fundamental field to describe the communicating parties and therefore essential information to describe a flow.
- **"sp"** - Source Port: The source port of the flow. In case of an outgoing flow this is the port of the user´s router where the traffic is coming from. In case of incoming flow, this is the Port on which the server responds to the client. This data field is considered to be highly important for detecting NAT, because Routers that perform dynamic NAT are considered to just increase the port number by one in case two or more clients in the private net want to communicate via the same Port. This is done to be able to still be able to distinguish the traffic related to a particular client. This behavior will generate clusters of just increasing port numbers in the flow records which are recognizable and therefore usable for detecting NAT.
- **"dp"** - Destination Port: The destination port of the flow. In case of an incoming flow this is the port of the user´s router where the traffic is directed to. In case of an outgoing flow, this is the port of the server where the traffic is directed to. This data field is considered to be highly important for detecting NAT, too. The same argumentation as for the source port also holds for this field.
- **"flg"** - This data field shows an aggregation of all TCP flags that occurred during the aggregation of the flow record. As described in the related work section, others proposed ideas to use the TCP-Syn flags as a feature for detecting NAT. The problem here is that the size of the Syn-packets is different for specific operating systems, but while NetFlow only provides aggregated data it can be difficult or even impossible to detect these small differences in the size of Syn-packets or in this case to recognize the differences in the payload size of the related flows. But nevertheless it is indicated which flag(s) were present during the aggregation of the flow. This field is just expected to appear during the setup of a connection.

It is important to note that the source ports can be affected in many ways. Because of this it appears in all the categories. It can, for example, be affected by the depending service the user consumes. The operating system can choose the port to communicate with a server after setting up the connection and routers can change the port number if there are several computers that try to establish connections on the same port. This in particular becomes very interesting with routers used in the private sector, because they often use a very simple approach to deal with these "collisions". Often, they simply increase the port number.

This behavior will be a part of later investigations, because it can probably be used as a feature to detect NAT. The basic assumption is, that the more computers/users are behind a

NAT-gateway, the more of such collisions will happen and there will be clusters of port numbers recognizable.

Affected by the user behavior:

- **"ts"** - Time Start: This is the time stamp for the start of the flow. This data field is of course highly related to the behavior of the user.
- **"te"** - Time End: This is the time stamp for the end of the flow. Like the start time, the end time also provides valuable information about the user behavior, because with this data field it is possible to calculate the duration of a flow.
- **"td"** - Duration: The flows duration. This can be used to see if the user just accesses any Web-resources for a very short time. For example a download of a huge field should be distinguishable from random browsing the Web.
- **"sa"** - Source Address: The IP address where the traffic comes from. In case of outgoing traffic, this address will always be the same. In case of incoming traffic, it is the server's IP address. In case of an incoming flow, it depends on the service the user(s) consume(s). To be able to describe the consumed services by one IP address at a certain time is very important, because some services are unlikely to be consumed multiple times in parallel by just a single user. For example it is unlikely that a single user consumes several Web-radio streams at a time.
- **"da"** - Destination Address: The IP address where the traffic goes to. In case of incoming traffic this address will always be the same. In case of outgoing traffic it is the server's IP address. Like the source IP address, this depends on the service the user(s) consume(s), in case of an outgoing flow. The same argumentation holds as for the source address, just considering the outgoing flows.
- **"sp"** - Source Port: This is also a data field highly affected by the user, because it depends on the service the user consumes. It could tell for example if well-known services are consumed that are related to fixed port numbers, such as for example Email-Services (POP3,IMAP, SMTP).
- **"dp"** - Destination Port: This data field is highly affected by the user behavior, because it is closely related to the service(s) that are consumed by the user(s). Same argumentation holds as for the source port.
- **"pr"** - Protocol: Indicates the protocol that is used during a flow. This data field describes for example whether UDP or TCP is used.
- **"flg"** - TCP flag(s): This field can also be affected by the user. For example, if the user tries to set up TCP connections, then TCP flags like "Syn" or "Ack" will also appear in the flow records. Additionally, these flags provide information which communicating party closed the connection. Thus this field also represents the users' behavior, but it can be hard to distinguish single-user from multi-user behavior, by using this data field.
- **"ipkt"** - Incoming Packet: Number of incoming packets during the aggregation of the flow. This data field highly depends on the users' behavior and the consumed services and the activity of the user(s). Basic assumption: more users will normally (on average) transfer more packets than a single user. The threshold for this is of course very fuzzy.
- **"opkt"** - Outgoing Packets: describes the number of outgoing packets during the aggregation of the flow. It highly depends on the users' behavior and the consumed services. As previously described, more users are assumed to create more traffic and therefore transmit more packets.

- **"ibyt"** - Incoming Bytes: The amount of incoming data while aggregating the flow. Also highly depends on users' behavior and the consumed services. The same assumption holds as with incoming packets.
- **"obyt"** - Outgoing Bytes: Amount of outgoing data during flow aggregation. It highly depends on users' behavior and the consumed services. The same assumption holds as with outgoing packets.

After this theoretical discussion of the usefulness of each of the data fields an analysis of these data fields derived from a simulation follows in the next section.

5.2 Simulated NetFlow data

The previous chapter provided a detailed, but theoretical overview over the data set and the different attributes it consists of. Some of these attributes, which describe the communication behaviour of the communicating parties, are changed on different levels of a connection. In contrast to this theoretical analysis, this section describes the effect that could be seen during the practical analysis of data, that was retrieved from the previously described simulation environment.

It makes sense to start with data from this simulation environment, bacause it offers the chance to trace back the source of the effect that shows up during the analysis. Furthermore, if certain effects are expected to show up, then this setup also provides the possibilty to change the setup in a way to provoke the effect artificially. This is useful to generally prove that it is possible that a certain expected effect can show up under certain conditions. Of course, this is always just the first step for further anlysis. Hence, if it is shown that an expected effect does not show up even in this artificial environment, then it does not make sense to search for it in a more relistic environment.

The first hypothesis that was analysed by using the methodology described before, was related to the assignment of source ports done by the NAT-gateway. This hypothesis was based on the fact, that due to the formal specification of the NAT overload process any NAT-gateway is able to change the port numbers of packets in general and is forced to change the source ports used on the outside interface(s) under certain circumstances. This might be for example the case, if there are "k" hosts on the inside and each of them at least tries to establish one connection "c" to the same destination by using the same source port "SP" as the other hosts. This would lead to the following situation:

$$\#Used\_SP < \#Connection\_Attempts \tag{5.1}$$

Since each of these hosts should be able to establish its connection to the desired destination and should not be limited in its communication by other hosts in the local network, it is necessary that the NAT-gateway translates the source port numbers to values that are currently not in use on its outside interface(s). Hence during the analysis based on a Cisco IOS 7200 image of version 12.2 an interesting effect was observed. The NAT-implementation of this image counted the source ports up only by one. This could be observed by first running one host at a time and observe the source ports it uses and then running two hosts and capturing the traffic from the router.

In the first source port usage of the single hosts looked like illustrated in figure 5.1.

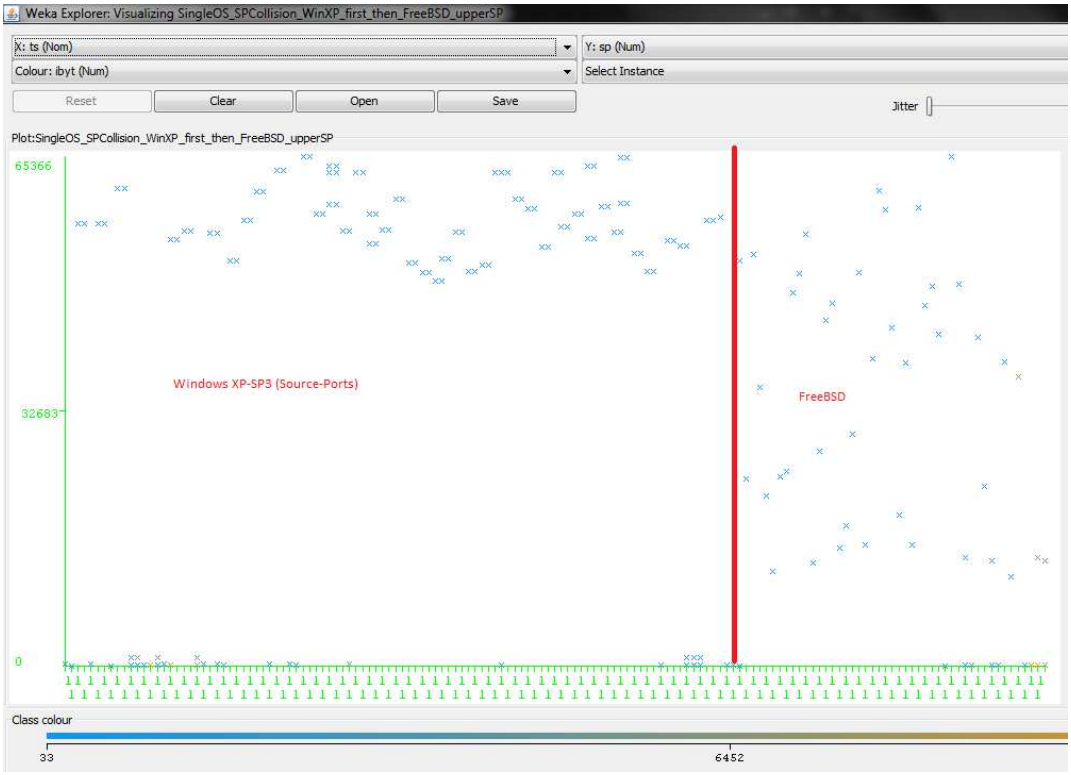


Figure 5.1: Port usage without NAT of Win XP and Free BSD host.

In the second step the source ports used by the NAT-gateway were observed. Figure 5.2 illustrates the observed Source Ports.

As stated before, the goal of this first setup was to generally prove that there are NAT-gatways that cause patterns in the source port usage that can be observed passively and remotely (e.g. by an administator or Internet service provider). In addtion, it is also possible to observe that different operatig system families choose their source ports for new connections from different ranges of the possible range of source ports.

The effect, described before, even seemed to appear independently of whether there were collisions in the usage of source ports on the inside interface or not. This new hypothesis could be proved with a second setup, where the setup was less artificial. For this setup the Simulation environment was connected to the Internet. This means that all the hosts had internet access and a realistic simulation of the communication behaviour was possible.

In this setup, 3 hosts were present and were running several services, such as E-mail clients, automatic antivirus updates, and a browser, that was in some hosts also automatically controlled by the previously described automated browser testing framework Selenium. One of them was running Ubuntu in version 12 and the other two host were running a Windows XP sp3 each.

Again, as before, the source ports used by each host were observed without NAT as well as the source ports used by NAT-Gateway. The same pattern of simply counting up the source port for each new connection could be observed again. Furtermore in these experiments it could be seen that the NAT-implementation seemed to have a default range in which it translates different protocols. For example, the counting for UDP seemed to repeatedly start at port-

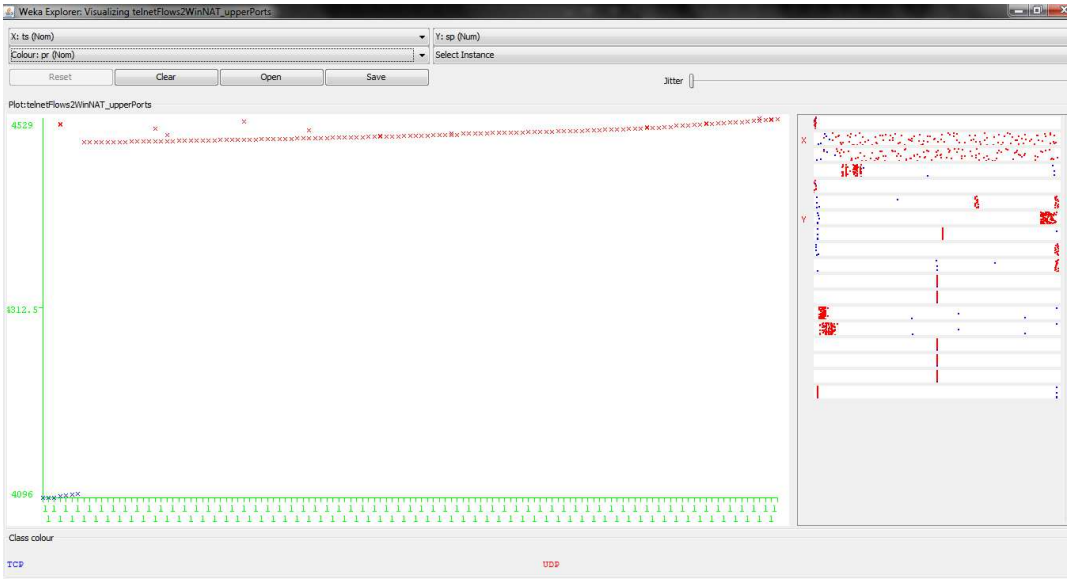


Figure 5.2: Two Win XP hosts connected via NAT-gateway.

number 4500 and the TCP source ports repeatedly were chosen starting from portnumber 4096.

The implementation might show this behaviour due to perforamnce reasons. This is due to the fact that this method of starting at a certain portnumber and then simply increasing by one, has very limited cumputational complexity, whereas pseudorandom source port assignment would have a higher computational complexity and since the computational power of a router is normally very limited, the developers might have choosen the simplest possible approach.

These observatios are of course only possible in the way of a black box test, since the actual source code of the NAT implementation used by Cisco is closed source software and therefore was not available.

To make sure that the patterns that were observed above have neither been introduced by the router hardware emulation software nor by the VirtualBox based Desktop emulation, a second setup, as described before was created.

This setup consisted of a Netgear router (WRN1000). Using this consumer router model also had the advantage that the two different use cases are met much more precisely, since private residential subcribers normally do not use business equipment. In addition, to this router 4 hosts were used to create connections and produce some traffic.

It can be seen in figure 5.3, that this router also shows the behaviour of assigning the source ports by incrasing them for most of the new connection attempts of the hosts on the inside. Hence, this time with real traffic the pattern was not as dominating as before. In this simulation the traffic was simultaniously captured at the hosts and the router. This enables to exactly see how the presence of the NAT-Gateway changes the used source ports from the hosts connected to the inside interface in relation to the outside interface.

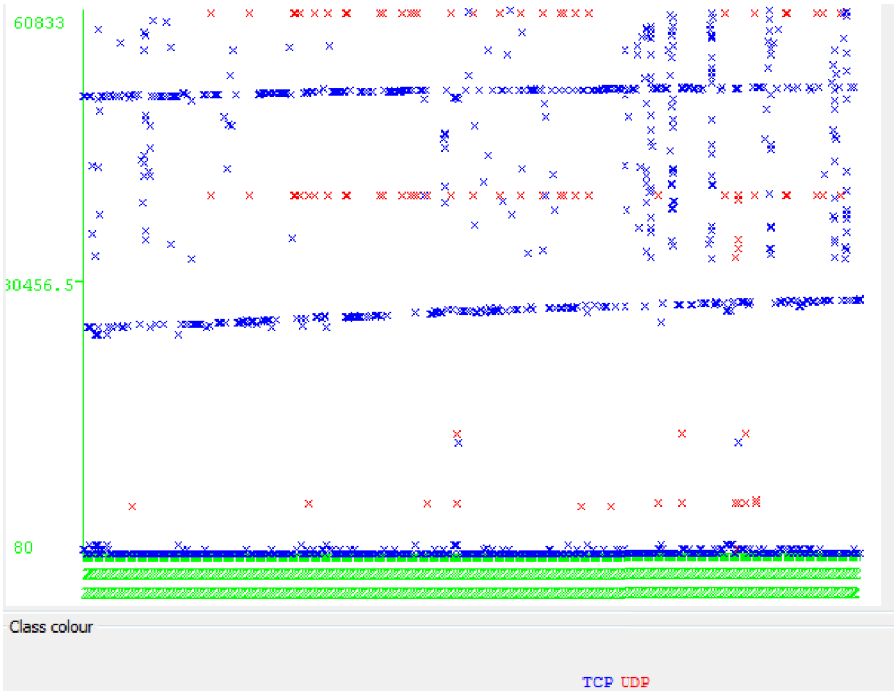


Figure 5.3: Source ports observed from Netgear router.

5.3 NetFlow from ISP

Within this thesis not only simulated data was analysed, but also a data set that was retrieved from a German ISP. The data set was retrieved from a business customer of the ISP, which was known to use NAT. As described in the fundamentals, such a data set has to meet various requirements, such as that it should be realistic and that the internal infrastructure should be well understood. The advantage of this data set was that it was very realistic since it was retrieved from a real network from the perspective of an ISP.

In contrast to the simulation, the disadvantage of this data set was that the information about the internals of the network were not as precise as in the simulation. Hence, there was at least some information, because as described in the previous chapter, about the extraction of sample data sets, in addition to the flow data the ARP-tables were exported every 15 minutes. This additional information provides information about the number of different hosts and when these hosts were active during each day.

As in the simulation environment, the first setp of the analysis was to plot the source ports over time. This again was done to see whether the linear behaviour that could be observed before in the simulation also appeared on this data set of real NATed traffic. An example plot of the ports used is presented in the figure [5.4]

As visible in the figure 5.4, this example for one day (24 hours on 10.09.2012) clearly shows the linear behaviour. The usage of the ports does not seem to be as purely linear as in both simulation setups before.

This might be caused by to the fact that there are more active hosts during the day and many TCP/IP stack implementations today use a so called "WAIT_STATE" (RFC 793), which is a timer for inactive connections. This timer is used to prevent ports that had been used shortly before to be assigned to new applications. This is important since some packets of a network data stream might take a different route and therefore arrive later, even after the

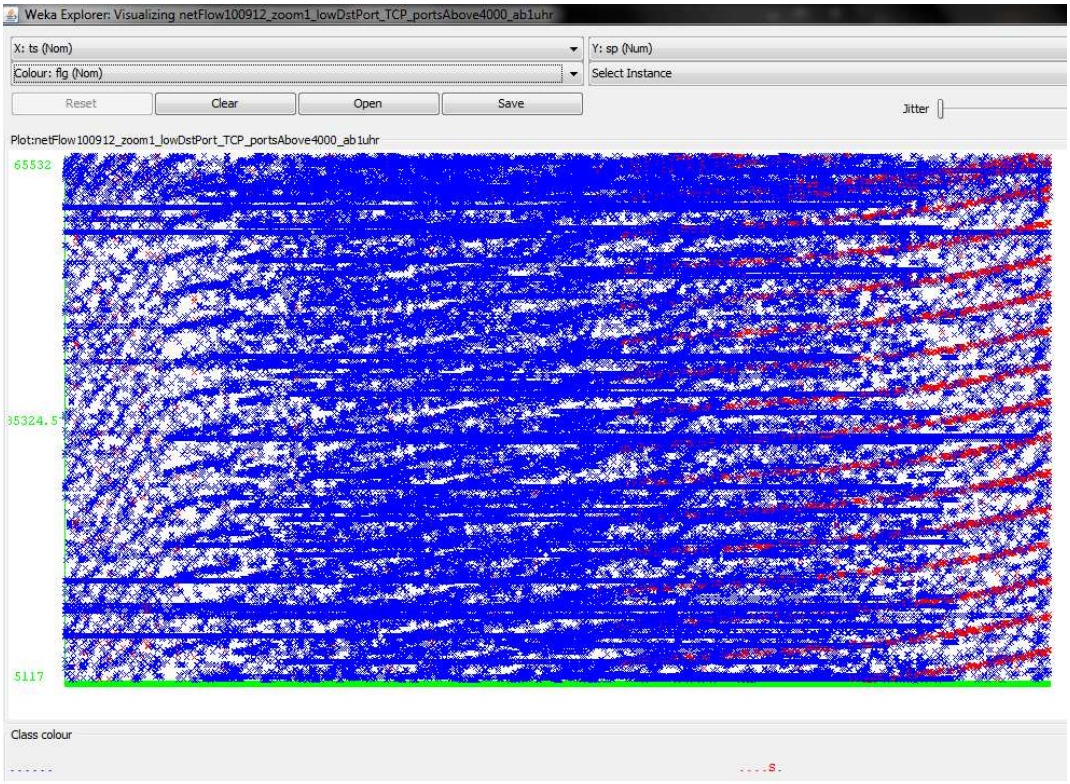


Figure 5.4: SourcePorts NATed ISP

connection is already closed. To prevent these packets to arrive at a network socket that is already assigned to a new application this timer is necessary.

A commonly used time frame that is used today, is 2 minutes. This is not mandatory for TCP/IP stacks. Especially in case of high network activity where not enough unused ports are available any more, a source port might also be reused earlier.

Knowing of this behaviour is of relevance, because the data set presented in figure 5.4 obviously shows high activity and a lot of new connections over the day, which might influence the port assignment behaviour. Since the number of port numbers in the NAT-gateway is limited also to the range [1024 to 65535] it might run into the previously mentioned situation, where it does not have enough unused free source ports for new connections. This is more likely to happen during noon, because as can be seen in the figure 5.4 this is the time where the network is most active at that day. On other days the highest activity was also around noon. This can be explaination for the fact that not all used source ports show up exactly in line, only upcounted by one.

Furthermore, it is also important to keep in mind that there are also a number of connections lasting over a relatively long time (even over hours). Such connections of course also "break" the linear port assignment pattern. In addition to this, it is also important to keep in mind that, as with the Cisco IOS image in the simulation, the internals of the implementation of the NAT-gateway can not be accessed, since the manufacturer keeps this software implementation closed.

This pattern is still of interest today, even though some manufacturers of professional business network equipment changed the assignment of the source ports to a pseudo random

assignment, because of security concerns with the predictable behaviour. In this context, it is also important to keep in mind that such hardware is normally used over many years before it is replaced by a newer product. Thus, many of these devices will still be used even though some manufacturers change this behaviour today.

Another interesting fact to point out with this data set is that the figure 5.4 also shows a lot of flows that only consist of SYN-Packets. These are highlighted in the figure by the red color. This is of interest, because as presented in the related work section, it has been shown by others that the size of SYN packets can be used for detecting different operating system families, e.g. Linux and Windows. So far this SYN based OS fingerprinting mehtod is normally performed packet by packet, but with NetFlow normally no information about a single packet is available. Thus, this measurement mehtod cannot be applied directly. However, a closer analysis of these flows - which are just based on SYN packets - showed that these flows are normally based on 2 to 4 of these SYN packets that appeared during the same second and therefore were aggregated into one flow. In addition, it is important to recall, that a flow not only contains information about the number of packets it is based on, but also the number of bytes that were transferred durring the aggregation of the flow.

This fact can be used to compute the average size of these "SYN-flows". Some randomly selected samples showed that the average size of these SYN-flows, very often corresponds to a known SYN-Size from the single packet analysis. The following figure 5.5 provides an example of such a randomly selected SYN-flow sample.

1	08:32:22	08:32:31	9000	74.49.136.110	54.126.63.79	18675	443	TCP	....S.	0	0	3	144
2	08:32:23	08:32:32	9052	74.49.136.110	54.126.63.79	43020	443	TCP	....S.	0	0	3	144
3	08:32:24	08:32:33	9048	74.49.136.110	54.126.63.79	18445	443	TCP	....S.	0	0	3	144

Figure 5.5: Example SYN-flow.

In this example the second last attribute of the flows show the number of outgoing packets and the last attribute shows the number of outgoing packets. When calculating the average based on these two attributes, an average size of 48 bytes per SYN packet can be derived. In the classical OS fingerprinting based on single packets this size is well known for Windows based systems. This is a very valuable result of the anallsis and is used in the next chapter to build a classifier based on this observation.

Another important observation during this analysis was that the number of opened connections, the amount of payload transferred within these packets was relatively high during the whole day. As discussed before, the amount of traffic transferrred normally reaches its peak around noon. Hence, there is at least some activity all the day. In contrast to this, single hosts are often just active for a few minutes or hours and then go offline again. This observation of different activities of single hosts in contrast to NATed subnetworks could be used to build classifiers for detecting NAT. In the next chapter of this thesis these observations will be used to build classifiers for a new NAT detection approach.

5.4 Extracted Features

Based on the observations of the simulated data set and the ISP data set, the following features are extracted and will be used within different classification approaches later in this thesis.

The first observation from the analysis above was that routers with built-in NAT-gateways - especially those which are designed for the private consumer market - often introduce a very

unique pattern of assigning source ports for new connections in a simple up counting fashion. This pattern could be found in the simulated ISO Router (IOS 7200, version 12.2) as well, but is regarding CISCO bug reports and documentation not indented and could not be found in current IOS images anymore.

The second observation was concerning the size of special flows that are solely based on SYN packets, which is normally indicated by the pattern "....S." in the TCP-flag field.

One very obvious observation was that more (active) Hosts and users will create more connections and in general will transfer more data. But even though NetFlow is a very high level statistical aggregation of the traffic transferred, it is still precise enough to distinguish different types of communication, such as flows related to certain network protocols, DNS-requests or connections related to E-mailing. The idea is to use all these possible attributes of a single flow and aggregate them over a certain fixed period of time. The result then is a description of how active a particular IP address was during a certain time frame and to which kind of traffic this activity belonged to. The table 5.1 provides an overview over all the features that are used in the next chapter for building classifiers.

Feature name	**Class**	**Description**
AvrgSYN	a	Average number of SYN packets
numInPkt	b	Number of incoming packets
numInByte	b	Number of incoming bytes
numOutPkt	b	Number of outgoing packets
numOutByte	b	Number of outgoing bytes
numMailCon	b	Number of flows related to EMailing
numSMTP	b	Number of flows indicating EMail sending
numTCP	b	Number of TCP related flows
numUDP	b	Number of UDP related flows
numDNSReq	b	Number of DNS requests
numSYN	b	Number of SYN packets
numRst	b	Number of RST packets
UpCuntingSrcPort	c	up counting of Source Porst

Table 5.1: Overview of feature set.

The table [5.1] lists all the features by their abbreviation and assigns a class label to each of them.

Class "a" is used for the Feature "AvrgSYN" because it is a unique feature that does not belong to any of the other classes. This feature is extracted by only filtering out outgoing flows with the SYN-flag set. These packets e.g. transmitted during a TCP connection establishment can be used as an indicator for different pperating systems. Thus class "a" will from now on be considered as a synonym to OS detection.

In contrast to this, class "b" classifies all features that describe the "behaviour of the connection". For example, the number of packets and bytes in the outgoing and incoming direction are highly related to the user(s) activity and the more users can be assumed to transfer more packets and bytes. The features "numMailCon" and "numSMTP" represent the behaviour of how much E-mail related flows could be found. In particular the feature "numSMTP" could be a good indicator for different users beeing active at a time, because it can be assumed that a

normal user does not send many E-mails in a short time (parallel) via SMTP of secure SMTP, because writing and hitting the send button would take him at least a few seconds. Each E-mail (sent via SMTP) causes a connection establishment to a certain SMTP server, which will then be present as a flow.

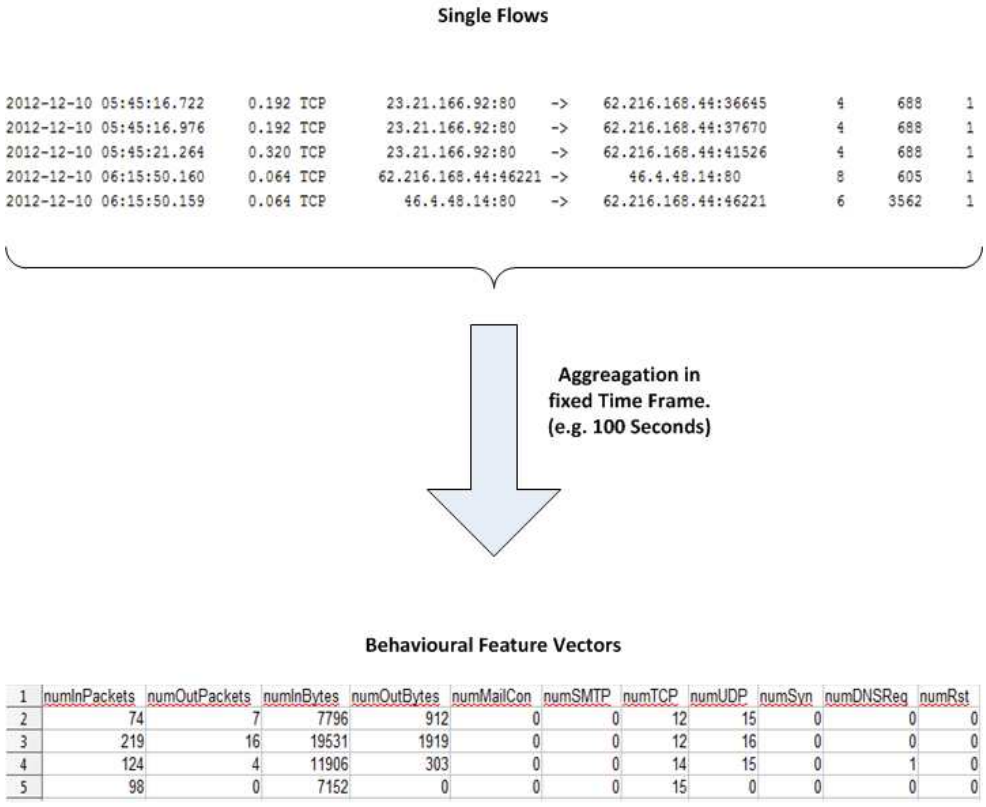
Another feature of class b is "numDNSReq", which represents the number of DNS-requests sent in a certain time. This feature is classified as behavioural, because - assuming a normal active users - the number of DNS-requests will be higher the more users are active and a single physical user normally will only create "a few" DNS-requests at a time. The previous observations of the data sets showed that this feature will often be present before a TCP connection is established, which is quite logical when for example assuming network traffic caused by browsing. Then the browser first looks up the IP address of a certain domain and then it creates a new TCP connection to it.

The "numSYN" feature represents the number of flows that contained a SYN-flag (maybe along with other TCP-flags). As mentioned before, the SYN-flag is an indicator for the number of connection establishments.

The "numRST" feature is the number of flows that had the Reset flag set within a certain time. This flag is set when a device rejecets the connection. Since NAT-gateways are often also advertised as a simple firewall refusing connections attempts from the ouside a higher number of flows having only this flag set could indicate that a NAT-gateway is present. This behaviour is likely to be present independentlyof a router in a private residential network or a professional router in a business envrionmnet is used as a NAT-gateway. It is classified under class "b" because it still describes the connection behaviour and could also occur for example because of closed applications, but it might also fit in group "c", if only malicious usage is considered.

The last feature "UPCountingSrcPort" is classified in class "c" because it describes some unique behaviour that is only introduced by the NAT-gateway itself. As described before, its presence is today most likely to appear with routers used in private residential networks. For this feature it is also important to mention that only the flows related to new connections are of relevance, because the source port will then stay the same for a connection once a source port was choosen while creating the connection. Therefore, the number of flows that have to be analyzed by this classifier can be decreased by applying a filter before the actual classification.

It was mentioned that the classification of the behavioural features will not be based on single flows, but on an aggregation of special information of flows that occured during a certain fixed time frame. For this approach the features of class will be aggregated and combined into a vector of features. In the following this will be called a feature vector. This principle is illustrated by figure [5.6]. Building classifiers based on these feature vectors will be done based on machine learning techniques since the feature vector is based on 11 attributes or expressed in a different way. It is 11 dimensional, which is a rather complex problem space to be solved. In the field of machine learning mehtods like Multi Layer Perceptrons or Support Vector Machines are designed to work on high dimensional feature spaces.



1	numInPackets	numOutPackets	numInBytes	numOutBytes	numMailCon	numSMTP	numTCP	numUDP	numSyn	numDNSReq	numRst
2	74	7	7796	912	0	0	12	15	0	0	0
3	219	16	19531	1919	0	0	12	16	0	0	0
4	124	4	11906	303	0	0	14	15	0	1	0
5	98	0	7152	0	0	0	15	0	0	0	0

Figure 5.6: Aggregation from single flows to feature vectors

CHAPTER 6

Classifiers

This chapter describes how the observations from the previous chapter have been transformed into different classifiers and a NAT detection system integrating these classifiers into a complete detection process.

6.1 Preliminaries

Previously within this thesis certain observations were made on different data sets. These observations can be used to classify a given unlabeled data set as either coming from a NATed IP address or not. In the previous chapter, a set of different features was described, which could be used to detect NATed network traffic. Different features were described to classify a set of flows to either the class "NATed" or "notNATed". The first feature described was based on average SYN-sizes, which depends on flows with special patterns in their flag field. The second classification approach was based on an observation based on sequences appearing while observing the used source ports of NATed flows over time.

In contrast to the previous two classifiers the third one is based on a more general observation of the communication behaviour of the user. This classifier is based on the 11 features, which were described before. These features were extracted from the flows of each data set (local network and ISP data) by observing and aggregating certain aspects (features) of the communication within a certain time frame. Then different machine learning algorithms were used to train classifiers first and then classify the actual traffic. The training was done on a subset of the data and the testing then was done on the remaining part. This way of splitting a limited dataset into a training and a test part is commonly used in the research community and was for example also applied in KDD-CUP [[17]].

Since three distinct classifiers were presented, each one with advantages and some limitations that will be decribed later in more detail. It is also necessary to combine these distinct decisions into one single descision in the end. Therefore, also a concept of decision fusion was

OS Familie	Typical SYN-Packet Size
Linux	60 Byte
Windows	48 Byte
OpenBSD	64 Byte
Solaris/Aix	44 Byte

Table 6.1: Overview common SYN sizes for operating system families.

developed.

The next section contains a more detailed description of the actual classifiers and a concept about how these different classifiers could be fuse into one single detection system.

6.2 Classifiers developed

In this section the different classifiers that were developed within this thesis are presented. This section focuses on the detection principle and algorithms of each classifier. The experimental results for each of the classifiers are presented in a later section. The classifiers are based on "Average SYN-flow sizes", "Source port sequences" and "User behaviour".

Average SYN-flow size

As mentioned earlier the first classifier is based on flows which are called SYN-flows within this thesis, because they only consist of one or more SYN packets that were aggregated into one flow. From the field of passive operating system fingerprinting, it is known that different operating systems have a different standard behaviour implemented in their TCP/IP protocol stack. As already described in the related work section there are many of these typical behaviours for different operating system families.

Most of them work on special data fields of the network packet headers, but NetFlow ignores most of these headers and aggregates the few data fields it considers over a certain fixed time. Because of this fact, most of the known techniques from the field of passive OS fingerprinting are not applicable on any NetFlow data set. There is one exception, which is based on the TCP SYN-flag. Different operating systems tend to set different option fields in their SYN packets and because of this behaviour SYN-packets coming from different operating systems normally have different sizes. The following table 6.1 provides some examples of typical SYN-sizes for different operating system families.

In the previous chapter "Analysis" it was discovered that there are special flows that appear in the data set, which have the special Pattern "....S." set in their flag field and often only contain a relatively low number of bytes. This pattern indicates that a SYN-packet was transferred within the connection that is described by this flow and because of this they will be called SYN flows in the rest of this thesis. As described in the previous, chapter the ISP dataset contained several of these SYN flows and some of these flows showed these typical sizes of different operating Systems directly, because they consisted of only one SYN-packet that was aggregated into one flow. In real life these flows, that are based on just one single SYN-packet, rarely occur. As described in the previous chapter there are numerous SYN-flows that also contain this pattern, but consist of several (e.g. 4) flows. Further analysis of these flows showed that very often, but not always, these flows are just aggregated of multiple SYN packets.

To consider this in the classification it is necessary to consider the number of transferred outgoing bytes in relation to the number of packets of each of these SYN-flows. Therefore, the first step of this classifier is to calculate the average packet size of a given SYN flow. In the case the flow is really just based on multiple SYN packets, then the derived average packet size will fit to a value of known packet sizes. This means that this classifier at minimum consists of the following parts: a filter, an average packet size calculation and a look up table with known SYN-packet sizes for different operating systems. Formally, this can be denoted as follows:

$$AvrgPktSize(SYNflow) = \frac{\#BytesOut}{\#Packets} \tag{6.1}$$

Where "AvrgPktSize" represents the function that returns the average size of the packets of a SYN-Flow. This calculation is based on "#BytesOut" representing the amount of outgoing bytes, which were transferred during the time this flow was aggregated. This value is set in relation to the number of packets that transferred these bytes. The following figure 6.1 illustrates these principal components of this approach as well as their interaction.

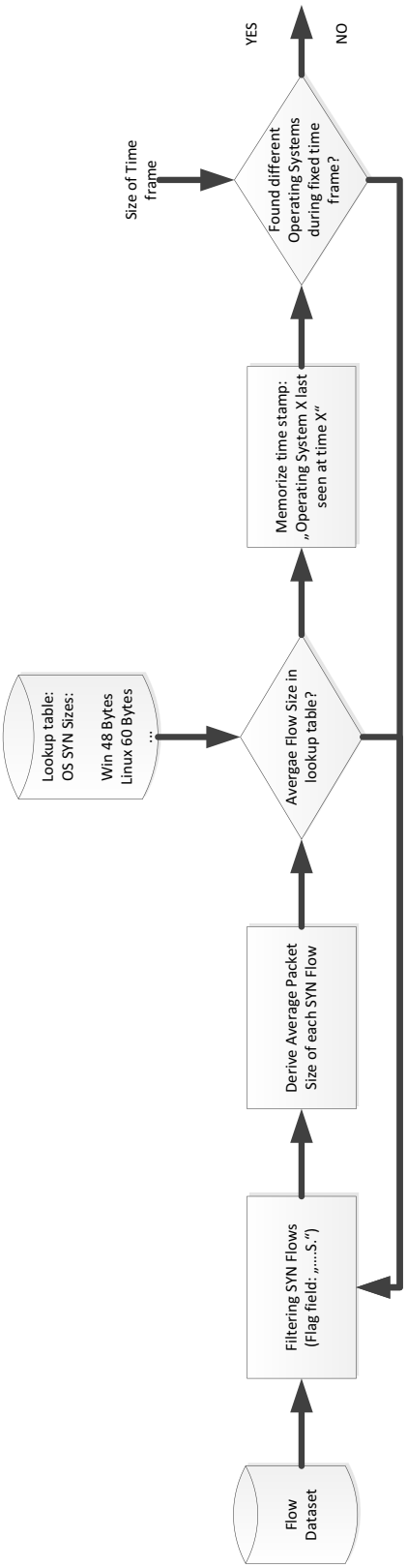


Figure 6.1: SYN flow based classification.

As shown in the figure 6.1, this classifier also considers a certain time frame. In case two or more different active operation systems were detected during that time frame then the classifier indicates that it detected the usage of NAT. It is also important to keep in mind that the first component, which is called flow data set, is assumed to be retrieved from one single IP address. Otherwise, this approach would not make sense. Moreover, it is also important to choose a proper time frame. One has to consider different elements for defining a proper time frame. First of all the time frame heavily depends on the intended use case.

Assuming the intended use case of this classifier is an enterprise network and IP addresses of the hosts can be assumed to be fixed (at least for a working day) and the aim is to detect the presence of unauthorized devices of the employees. Then it might be sufficient to choose a time frame of several hours or even a whole workday for this classifier.

Also, if the goal is to detect NAT in a network of an ISP, then time frames in the scale of hours might be sufficient in many cases. Hence, if this approach is assumed to be used within a real-time Intrusion Detection System or within an analysis of a still active botnet, then even time is very limited. It is obvious that the information of this classifier should be available within a timeframe of minutes. In the case of analyzing a still active botnet this approach could be helpful to more precisely determine the size of the botnet, because this approach is able to provide a lower bound of the size of a NAT. This information can then be used as one input for a more precise botnet size estimation.

The main disadvantage of this approach is that these SYN-flows appear only very rarely in normal traffic. In most cases, the presence of the SYN flows in the data set is caused by some kind of anomaly in the communication, such as misconfigured E-mail clients, portscans and also slow response or unavailabilitiy of an internet resource (high load or maintenance of Web server).

Nevertheless, this classification approach has several important advantages. The main advantage of this approach is that it is not only able to provide a binary decision whether NAT is used to share a certain IP address, but also it is able to reliably determine the lower bound of the network size behind the NAT-gateway.

Furthermore, it is important to consider that the previously mentioned disadvantage of the low number of appearances of these flows in normal traffic only limits this classifier's performance in case it is applied on normal traffic. Assuming the NAT detection should be done enhance the precision of estimating the size of a botnet, then abnormal traffic can be assumed to be the default case and the performance of this classifier might increase in some cases.

This gets clearer when assuming a use case where a botnet should be analyzed that is mainly used to send SPAM E-mails, and a certain IP address is identified or assumed to be a part or this botnet. In case there are multiple infected hosts behind this IP address and each one sends lots of E-mails during a short period of time, then it is much more likely to happen that not each of their connection attempts to SMTP servers are successful, which would cause the SYN-flows to show up. If in such a scenario the hosts run different operating systems then this classifier will detect these different SYN flows.

Source Port Sequences

Besides detecting NAT indirectly by detecting the presence of different hosts/operating systems, it is also possible under certain circumstances to detect the presence of a NAT-gateway directly by certain patterns. Some of these systems inject these patterns accidentally, because of a bad implementation of the NAT overload principle.

As already described in the fundamentals chapter, NAT overload is basically a combination of IP address and Port translation. During the analysis of the assignment of source ports by different operating systems a common pattern was observed from different NAT implementations. In case there are multiple hosts on the inside of a network that try to establish a connection to an outside resource on the same source port this causes a "collision" in the NAT-gateway. In order to solve this problem the NAT-gateway has to change at least one of the Source ports to distinct values. This means the NAT-gateway also has to memorize that it changed the port to apply these changes in the inverse direction for incoming packets.

Many NAT-gateways solve this problem in a very simple way, by just increasing by one the source port of the connection that is established slightly later. Since this can under certain circumstances be a security flaw, vendors of professional business equipment changed this behaviour in their devices to a pseudo random assignment of the source port numbers.

Since the source ports are only assigned during the establishment of new connections, it is possible to ignore all the following traffic or respectively all the following flows. The following flows of an established connection will use the same IP and port configuration. Flows that correspond to a connection attempt are for example DNS-requests or Flows that have a SYN-flag set. The following figure [6.2] shows an overview over the relevant parts of this classification approach.

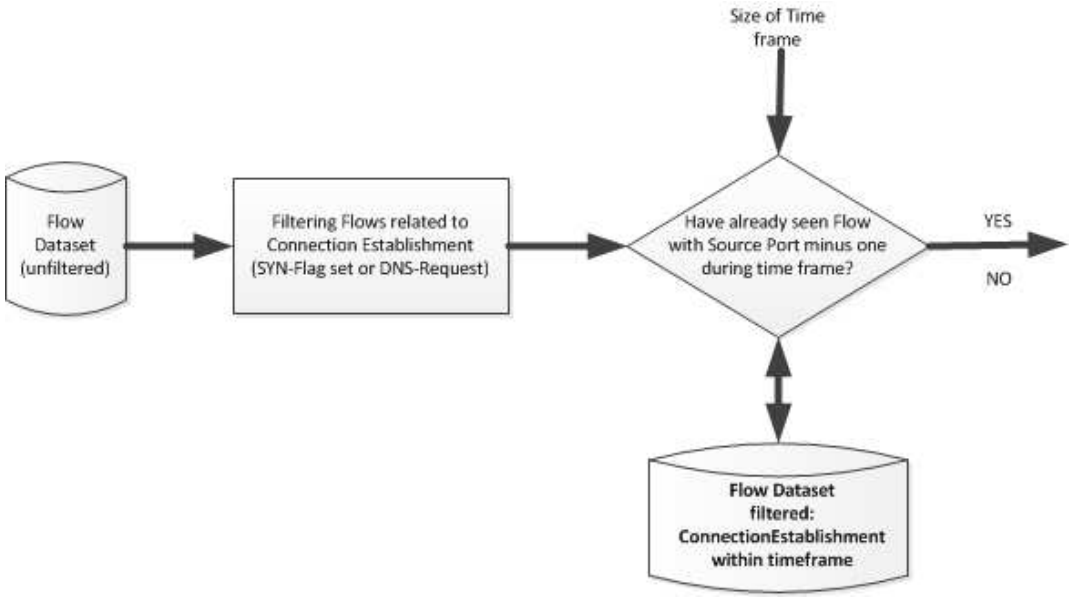


Figure 6.2: Main components soure port classifier.

As described in the Figure [6.2] after filtering out just the relevant part of the data set the next step is the actual classification process. This is mainly done by searching within a certain fixed time frame, if there was a previous flow with a source port number that is just one number lower. Even though the main idea of this approach might look fairly simple at a first glance, it is important to also keep in mind that the previously described method only works on ordered and complete data. But NetFlow data is normally transferred via UDP datagrams, which means there is no guarantee that all flows arrive at the flow collector component. Hence these flow datagrams contain other valuable information that can be used to extend the previously mentioned approach by some error detection and correction.

The following pseudo code illustrates how the classification algorithm works:

- A) Datastructure PortSequence:
  - timeStart
  - timeLast
  - firstSourcePort
  - lastSourcePort
- B) The Algorithm
  1. Initializion:
     - Define list of currently active port sequences.
     - Define list of sequences that could not be assigned to a port sequence, yet.
     - Initialize timer "currentTime" with timestamp of first flow.
     - Initialize threshold (eg. 120 sec).
     - Initialize cache flowSequencesLost.
     - Initialize lastSequenceNumberSeen.
     - Initialize DetectionThreshold.
  2. Read and analyse next flow (if any).
  3. If Src-Port of flow equals lastSourcePort +1 of any of the currently active port sequences.
     - Then: update timeLast, lastsourcePort and last SequenceNumber in that port sequence.
     - Else: Goto 4)
  4. Check, if flow with SourcePort deviation of 1 has been cached already.
     - Then:
       * Create new PortSequence in cache with timeStart = timeLast = currentTime;
       * firstSourcePort = lastSourcePort = Source Port of current flow;
       * lastSequenceNumber = SequenceNumber of this flow.
     - Else cache current flow.
  5. Remove any Portsequence from cache that satisfies (timeLast + threshold) <currentTime.
  6. Remove any Flow from cache that statisfies (timeEnd + threshold) >current Time.
  7. If Number of Flows in cache >DetectionThreshold
     - Then: NAT was found
     - Else: Repeat steps 2 til 6 until NAT was found

The pseudo code can be grouped into several parts. For example all initialization work is done in step 1 and step 2 defines a loop to read new flows until all flows have been processed. The actual detection of Portsequences is done in Setp 3 and 4. In steps 5 and 6 the caches are sanitized to prevent overflows, and to reduce the number of flows all new flows have to be checked against. These two points are in particular of interest, if the detection approach should be applied on provider networks. Finally in Step 7 the abort condition is checked. This abort condition is based on a threshold in practice that defines the number of sequences that has to be found until the algorithm is convinced that a NAT-gateway is present.

Two data fields of these flow datagrams are of particular interest for such an approach, namely the Sequencenumber of the flows and the number of flows contained within the datagram. The sequence number is a value that is increased by the flow exporting device and is increased for each flow that this device exported. Normally, this value corresponds to the total number of flows that this flow exporter component has already exported since it was started.

By directly capturing the UDP datagrams from the network interface it is possible to read these data fields as well. If the detection of missing values is of interest, then this sequence number has to be monitored all the time. In case a missing number is found, then for example flows that arrived earlier and later during a certain fixed time frame could be searched for further analysis. Assuming there have been flows indicating connection establishments with Source Ports of similar very close Source Ports, then the missing flow(s) could possibly fit into this gap and the sequence can be extended correspondingly.

Furthermore, directly capturing the flow datagrams could also be of interest in case a more precise time estimation would be necessary. This could be done, because the datagrams contain a time stamp in Unix time format with millisecond precision, while most flow sampling tools (e.g. nfcapd and nfdump) limit the precision to a scale of seconds, in their standard configuration. Since for this thesis the ISP data sets were available only in sampled format this error correction was not necessary. Furthermore, also in the case of the simulated data set it was not necessary to implement this correction approach, because in this configuration no packets could get lost. Hence, in a practical application it might be useful to consider this approach as well.

As seen with the previous classifier a more "active"/"connective" host will support the detection rate of this classifier. For example, if a botnet is used for a DdoS Attack against any target, then it will try to create numerous connections in a relatively short period of time. This behaviour increases the probability that collisions occur with used source ports of other hosts in the network, especially if these hosts are part of the same botnet and are attacking the same target e.g. a Web server at the same time. Therefore, this classifier is like the previous one not only very robust against abnormal/malicious traffic, but also it performs even better in this abnormal domain.

Nevertheless this classifier has the disadvantage that with current hardware - especially in enterprise networks - this pattern becomes more and more unlikely to show up. Furthermore, with a low number of devices and low activity the probability for these "collisions" in source port number is very low. This also reduces the probability that these port sequences can be seen.

"Behavioural Detection"

The previously described approaches work on different levels in the structure of a NATed network. While the port sequence classification approach attacks the classification problem on the level of the NAT-gateway, the SYN flow based approach attacked the problem on the level of the host or the operating system respectively. Both approaches were developed based on domain specific expert knowledge that originates from special observed patterns.

In contrast to these approaches, the classification approach that is presented within this section is based on a number of assumptions that are practically transformed into a feature set consisting of 11 features. The feature set has already been described in the previous chapter in more detail. In general, these features can be seen as a very high level statistical description of the communication behaviour during a certain fixed time frame.

After transforming the data set consisting of multiple flows into a set of feature vectors

that are an aggregation of different fields of multiple flows during a certain fixed time frame, these feature vectors are used to train special machine learning algorithms. The main difference to the previous detection approaches is that the model for detection is not created by an expert, but by using special machine learning algorithms, which learn the detection model autonomously from a set of training data.

A proper training set contains positive as well as negative examples. In the remaining part of this thesis positive means that the feature vectors were retrieved from network traffic originating from a NAT gateway. In contrast to this the term negative is used to describe, that the feature vectors were retrieved from a source without a NAT in between, e.g. directly from a single host.

For the actual training, a mixed set of features was created. Mixed in this context means that a label was assigned to each of the feature vectors, saying either whether it was retrieved from a NAT-gateway or not. Then different classifiers based on different machine learning techniques were trained on this mixed set. The output of this process, was a model that could be applied on new - also previously unseen- feature vectors of which the class they belong to is unknown.

For analyzing the detection rate of the classifier the number of correctly classified samples was set in relation to the total number of provided samples. The number of classified samples is determined by adding the number of samples classified correctly as positive (True Positive or short TP) and the number of samples correctly classified as negative (True Negative or short TN). The detection rate is then given in per cent. Formally the detection rate is denoted as:

$$Detectionrate(in\%) = \frac{(\#TP + \#TN)}{\#Samplestotal} \times 100 \tag{6.2}$$

The evaluation of these classifiers follows in the next section "Experimental Results" of this chapter.

6.3 Experimental Results

Since three distinct detection approaches have been presented in the chapters before as well as a method to combine all these approaches into one single system, by using an ensemble method, it is also essential to validate the detection accuracy of the distinct parts as well as the ensemble. The classifiers have been applied on both the simulated data sets and the data sets of the real ISP.

The methodology and the actual results are described in the following sections of this chapter. The focus of the experiments is more on the real ISP data since this data represents the reality and is therefore much more representative. As described before, the big advantage of the simulation environment was the possibility to figure out the influences of the different components involved.

6.3.1 Test Methodology

After implementing the classifiers described before, they were tested on two different data sets. These data sets represent two distinct classes of networks. The first data set was retrieved from a small local network, which represents a typical private residential customer.

Furthermore, a second data set retrieved from the network of a customer of a German ISP was used as well. This data set represented two different things. First, it represents the view of an ISP on a business network and second it represents a business customer/network, which can normally be assumed to have a different network usage than a private residential customer.

In general, the process of retrieving feature vectors was the same on flows retrieved from private equipment and the ISP data set. The only difference was that for the ISP data set first only positive (flows known to be retrieved from a NAT-gateway) were available. Since at later point in time also negative flow samples have been available these were added to make both the training and testing scenario more realistic.

It is important to keep in mind that the port sequence classifier and the SYN-flows based classifier work directly on the flows. Hence the user behaviour based classifier, which is based on machine learning algorithms, needs a more complex preprocessing of the flows in order to apply the actual algorithms. For this classification approach the methodology was as follows.

As described before, in the beginning in both scenarios there were two flow data sets. One containing NATed flow samples and the other one containing not NATed flow samples. Each of these data sets was then split into two parts: a training portion and a test portion. The result were four subsets of flows. Each of these subsets then was converted into feature vectors. In the subsequent step, to all of these feature vectors a label was assigned, indicating whether the feature vector was based on NATed or non NATed network traffic. Then the two training portions were combined to a mixed labeled training set and the remaining two portions to a mixed labeled testing set. The last step then was to train and test the different detection algorithms on these data sets.

Regarding the feature vector generation, it is important to mention that each feature vectors was aggregated based on the flows that were present during a certain fixed time frame. The algorithms have been tested on time frames of different sizes, such as 60 sec, 80 sec, 100sec 120sec

After creating these feature vectors, different machine learning algorithms were used for building classifiers for such a data set. Such a classifier needs data, which it is trained on and data, which then should be classified. In this case, this second data set was retrieved from the same data set as the training set and it was used for testing the classifier's performance.

It is important to recall the definition of the detection rate from the fundamentals chapter. All detection rates listed later in this chapter follow the following definition:

$$Detectonrate = \frac{\#CorrectInstances}{\#Instancestotal} \times 100 \tag{6.3}$$

Where number of correctly classified instances in relation to the number of instances that have been classified in total. The result then is the detection rate in per cent.

6.3.2 Results of SYN approach

As described before, the SYN-flow based detection of NAT based approach is based on special flows that only consist of TCP SYN-packets, which are aggregated to one flow. These flows appear very seldom and never appeared in the simulated setup and the small local network setup. Therefore, no further results for these setups can be presented here.

Day	Num SYN-Flows total	Num Linux	Num Windows	Num Other
Day 1	540	539	0	1
Day 2	550	550	0	0
Day 3	573	550	0	23
Day 4	528	528	0	0
Day 5	561	560	0	1
Day 6	543	543	0	0
Day 7	547	547	0	0
Day 8	5665	559	50064	42

Table 6.2: Overview of SYN flows in the ISP data set.

Day	Number of sequences	Flows being part of sequence	Flows total
Day 1	18	46	186173
Day 2	21	52	212854
Day 3	50	126	184147
Day 4	20	50	167922
Day 5	21	51	191408
Day 6	1	2	81582
Day 7	0	0	81314
Day 8	24	69	190011

Table 6.3: Overview of source port sequences in positive ISP data set.

Because of this very rare presence of these flows, it is obvious that this classifier will achieve only a very low detection rate. The data set retrieved from the ISP contains 5665 of these flows within a period of 8 days. Seeing this in relation to the 2.567.766 flows in total during this time, then these flows only represent approx. 0,22% of the total amount of flows. Depending on the time frame chosen for this detection approach, even from this subset just a few of these flows can be used for detecting NAT. The following table 6.2 lists the "detection rates" of this approach.

The table 6.2 above clearly shows that this classifier needs to be applied within a time frame of days or even weeks in reality, but it also shows that these flows and differences appear in reality. Day 8 seems to be abnormally high, but this classifier is still able to detect the different operating systems. It even increases its precision.

6.3.3 Results of Src-Port assignment approach

As described above, this classifier detects sequences in the used source ports. This classifier was applied on flow samples of 8 days and listed for each day how many sequences were found, as well as the number of flows that are part of such a sequence. The following table 6.3 shows these results.

The number of sequences and flows being part of a sequence might look low at a first glance but it is important that flow export was done in 5 minute slots by the provider. Because of

Host	Number of sequences (1 day)	Flows beeing part of sequence	Flows total
Host_1	0	0	504
Host_2	0	0	153
Host_3	0	0	250
Host_4	5	10	24740
Host_5	8	16	2290
Host_6	38	76	314
Host_7	93	46	2541
Host_8	36	72	373
Host_9	0	0	573

Table 6.4: Overview of source port sequences in negative samples.

this, not all flows that are part of a sequence in reality can be found in this data set. For the same reason also 5 minutes were used as the size of the time frame in which a flow is still considered to belong to the same sequence. Furthermore, on this data set the possibility to detect lost flows does not exist, because this classifier would need the flow datagrams to do that.

After applying it on positive (samples retrieved from a NATed network) this classifier was also tested against a set of known to be not NATed samples retrieved from single hosts. The results of this test are listed in the table 6.4 down below.

6.3.4 Results of behaviour based approach

Like the previous detection approaches, also the user behaviour based detection approach was tested on two different test data sets representing two different scenarios. The first data set represents a private residential network and the second data set represents an enterprise network seen from the perspective of an ISP.

Private residential network

The first data set was retrieved from a small local network consisting of 4 Hosts (2 Windows and 2 Linux) which was especially set up for this test. The flows of these hosts were sampled over a period round about 100 minutes. These flows were captured on both the NAT-gateway itself to provide positive samples as well as on each of the hosts to provide negative samples. From each of these different sample sets the previous ly described 11 feature vectors were retrieved. In addition, a label was assigned to each of the retrieved feature vectors indicating whether it was retrieved from NATed samples or not.

All labeled feature vectors were combined into one mixed set of feature vectors. This mixed data set was then used for training classifiers based on different machine learning algorithms. Since the data set was rather small no splitting of the set was used. For testing the classifiers performance the common 10 fold cross validation was used.

As mentioned before, different machine learning methods were used to build models for classifying the feature vectors either as "NATed" or "not NATEd". The table 6.5 provides an overview over the performance of different algorithms that were used. The used sample data

Algorithm	Detection rate
DecisionTree	78,20%
RandomForest	75.61%
SVM	73,18%
NaiveBayes	74,74%
MLP	75,78%

Table 6.5: Overview of detection rates of different algorithms.

set contained both NATed and non NATed flows from which the feature vector was retrieved in a 120 second aggregation.

The previously described results are also illustrated in figure 6.3

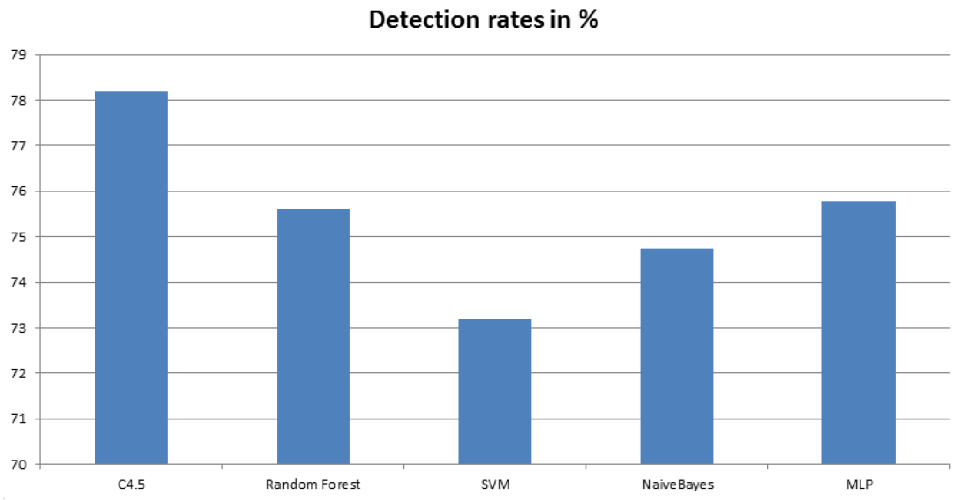


Figure 6.3: Overview of detection rates on local network setup.

It is important to mention that the data set was based on 418 feature vectors representing NATed traffic and 161 feature Vectors representing non NATed traffic. This imbalance might affect the model a little bit. It is important to mention that the detection rates are quite good, bearing in mind that the difference between single user and multi user can be expected to be less strong then for example on the ISP data set, that is presented in the following. For this setup it would not be surprising if the detection rates might even be a little bit higher than in this local network setup.

Provider data

The Provider data set was based on two different sample sets, positive ("NATed") flow samples as well as negative (not "NATed") flow samples. These flows were retrieved from two different business customers. When applying such a system in reality it is essential to train the classifier on samples that fit to the data which should be classified later on. Thus a subset of this data was retrieved for training and the rest for testing, as it was done for example in KDD Cup [[17]] before.

Algorithm	60Sec	80sec	100sec	120sec
DecisionTree	92.6%	93.41%	93.44%	93.13%
RandomForest	92.6%	93.36%	93.04%	92.88%
SVM	92.7	93.04%	93.70%	93,67%
NaiveBayes	24.1%	24.7%	25.12%	25.53%
MLP	90%	88.35%	88.90%	89.90%

Table 6.6: Overview of detection rates of different algorithms.

As mentioned before, it is also important to keep in mind that the feature vectors are based on certain data fields of all flows that occurred during a certain fixed time frame. The best size for this time frame was determined empirically. A good starting point for this analysis was provided in [[20]], where the authors propose that they achieved their best results in a similar approach with a timeframe of 100 seconds. The following table [6.6] list the results that could be achieved with different algorithms based on time frames from 60 seconds up to 120 seconds.

The training samples were taken from two days and the tests have were done with 6 of the following days. For negative training samples flows retrieved from 14 single hosts, retrieved from a different business customer were used. With this approach different classifiers were trained. The best result we could achieve with this method was a detection rate of 93,7%. This result was achieved by using a SVM with a kernel of size 7. The table 6.6 provides an overview of the detection rates of different algorithms that were used.

The figure 6.4 illustrates the change in the detection rate of the C4.5 decision tree algorithm with different time frames.

The figure 6.4 clearly shows that the time frame of 100 seconds provides the best detection rate for the C4.5 algorithm. As seen in the previous table, this also holds true for most of the other algorithms, as well.

The following chart 6.5 illustrates a comparison between the different algorithms used with the time frame of 100 seconds.

All classifiers were applied on the exactly the same training- and test sets. Even though the SVM achieved the best detection rate, this does not mean that it would by default be the best choice in practice. This is because of the fact that the table shows that also other approaches reach almost equal detection rates and in practice other factors are also of high relevance. Especially when it comes to the speed of a classifier the SVM performed the worst on this test set. Also the time consumption for training the classifier might be of relevance in practice.

All classifiers were trained on the same mixed training feature vectors, which were aggregated in time frames of 100 seconds and contained 79047 feature vectors. This data was retrieved from two different business customers of a German ISP and contained traffic from two days.

The mixed test set for 100 second aggregation of the feature vectors contained 405.452 samples of both negative and positive examples retrieved from two different business customers of a German ISP over 6 days. One of the customers was running a NAT-Gateway and the

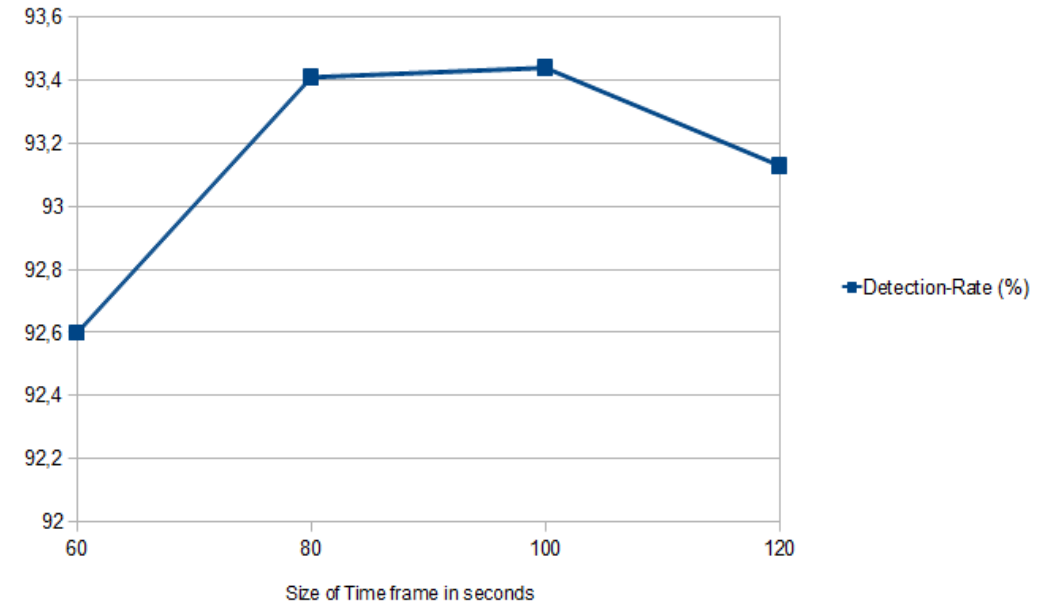


Figure 6.4: Detection Rate C4.5 with different time frames.

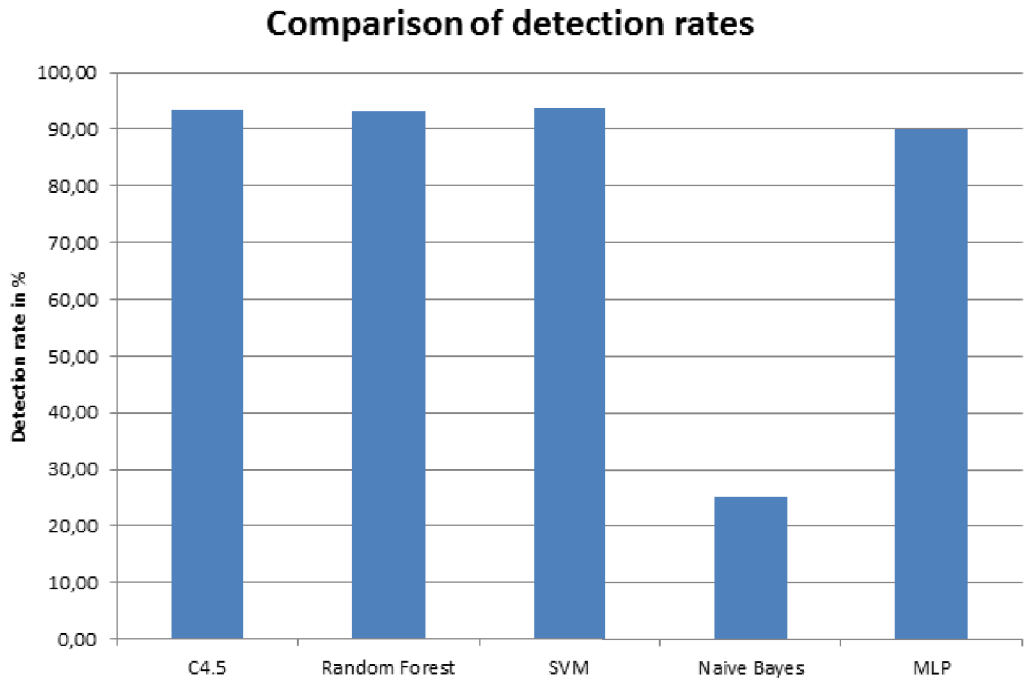


Figure 6.5: Comparison of detection rates at 100 second aggregation

Algorithm	Time to build model (sec)	Time to classify the test set (sec)
C4.5	7,32	2,22
Random Forest	11,79	3,63
SVM	1366,99	1371,46
Naive Bayes	0,47	3,16
MLP	391,19	2,59

Table 6.7: Comparison: Time taken to build model and testing.

negative samples were retrieved from 14 distinct hosts of another business customer. The following table 6.8 presents a comparison of the time taken to build the model and to classify the supplied test set for each classifier.

When visualizing these values, it becomes very clear that the SVM would be the worst choice to make from the runtime perspective. This comparison is illustrated in figure 6.6.

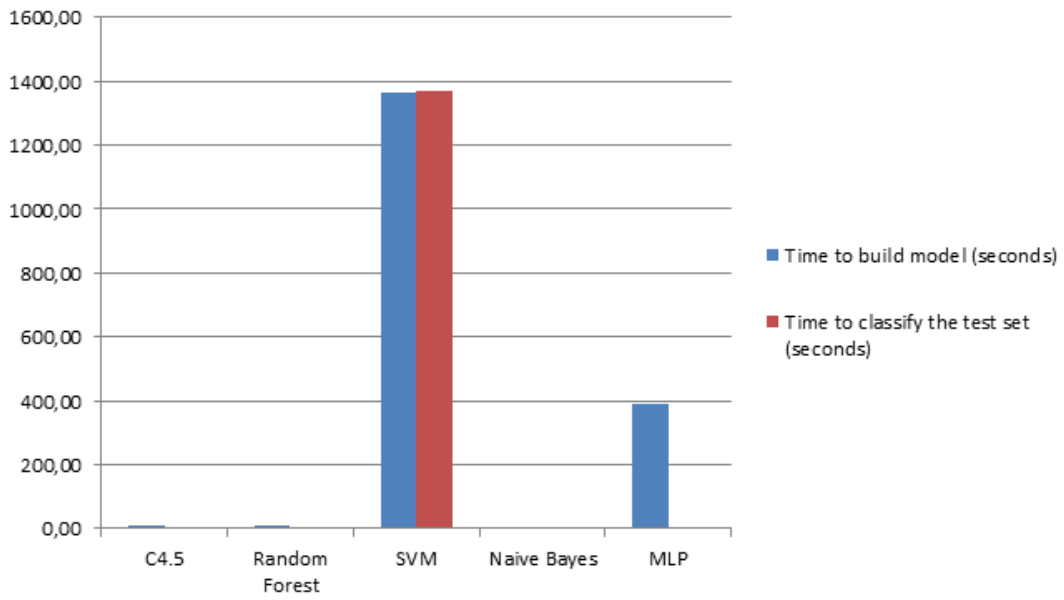


Figure 6.6: Comparison: Time taken to build model and runtime on test set.

Considering the detection rates, the time to build the model and the time to classify a large data set, the C4.5 decision tree based algorithm performs best and would be the best choice. A further advantage of this machine learning technique is that its model is more or less human readable and understandable. On such a high dimensional data set the human readability is practically very limited. Nevertheless, it at least reveals its internal model. The Multilayer Perceptron (MLP) also achieved quite a good detection rate and was very fast when it was applied in practice. It suffers from two drawbacks, namely the first drawback is that it takes relatively long to build its model and it can be seen as a black box. This means it does not reveal anything about its internal model. The drawback when it comes to runtime might not be relevant in many cases, but not having any possibility to analyze might be the strongest drawback of this method.

After performing this analysis also the port sequence classifier was applied on this data set as well. The detailed listing of this test has already been presented before. Since it showed a lot of sequences in the data set that the ISP provided as "non-NATed" flows, the ISP asked their customer to provide more details about his network to recheck that the traffic is really coming from "non-NATed" sources.

The outcome of this check was that indeed this data set also contained traffic from one device definitely acting as a NAT-gateway. Furthermore, two other routing devices, of which it is not known whether they perform NAT or not. Because of this, a few of the feature vectors used for training before were labeled in a wrong way. Since the data was used for both training and testing the machine learning based classifier, this might have reduced the detection rate of the classifiers. Therefore, the three devices were removed from the data set. This means the feature vectors for the negative ("not NATed") samples were retrieved from only 11 hosts instead of 14 hosts, as it was done before.

The training was then performed again on the feature vectors from the first two days. This training set was based on 1.370.880 non-NATed flows which were aggregated to 11.125 to 13.811 feature vectors depending on the time frame used for the aggregation. For the positive case the same samples were used as before, this means that it contained 63.108 to 64.627 feature vectors that are based on 425.084 flows that have been retrieved from two days known to be NATed traffic.

The figure 6.7 illustrates the proportions of NATed and non-NATed flows as well as the proportion of the resulting negative and positive feature vectors that were used for training the different classification algorithms.

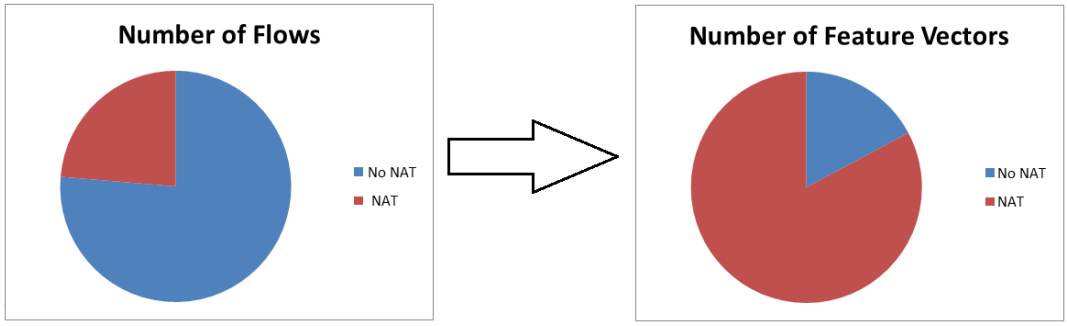


Figure 6.7: Proportions of data sets for training.

Moreover the test set contained 2.550.619 NATed flows which were aggregated into 360.729 to 369.525 feature vectors, based on the traffic retrieved from 6 days. The negative samples were retrieved from traffic of 11 hosts captured also over 6 days. This then created 2.284.800 flows, which resulted in between 30.736 and 34.994 feature vectors as negative samples for the algorithms, again depending on the chosen time frame for the aggregation of the feature vectors. It is obvious that the number of flows provided for both cases is rather balanced, and the resulting set of Feature Vectors is not. This becomes even more obvious in the following chart.

Comparing the number of flows of the positive and negative samples used for training or testing it is noticeable that the positive samples are rather over-represented again, hence the negative samples are based on more flows. An analysis of this phenomenon showed that this is due to the fact that single hosts seem just to be active during a few minutes/hours of a day. This also means that feature vectors were retrieved during these few minutes/hours. Also a

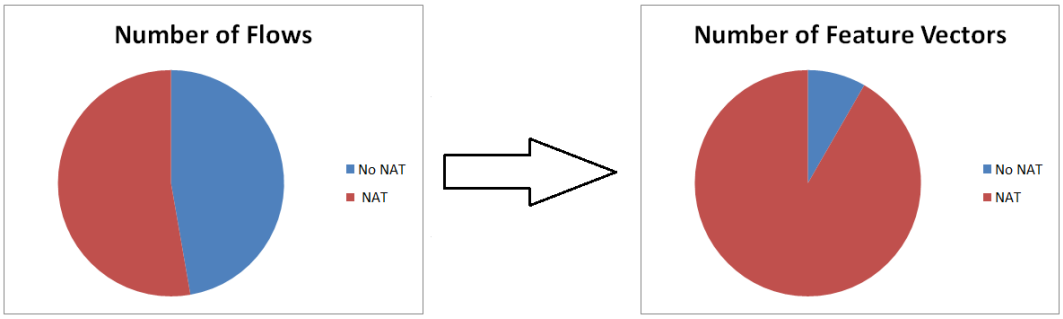


Figure 6.8: Proportions of dataset used for testing.

Type of samples	Number of flows	Number of feature vectors (rounded avrg.)
Pos. Training	425.084	63955
Neg. Training	1.370.880	12712
Pos. Test	2.550.619	365447
Neg. Test	2.284.800	32900

Table 6.8: Comparision of times taken to build model and testing.

few minutes of inactivity could reduce the number of flows retrieved.

In contrast to this a NAT that "hides" a lager network (a few dozen hosts) is very likely to have at least a little bit of traffic transferred all the time. In fact, this could be seen when analyzing the data set. The NATed network was active almost completely 24 hours a day and just creating one flow every 60 seconds or up to 120 seconds would, depending on the timeframe that is used, result in a feature vector that is created.

The table 6.8 provides an overview over the proportions of the training and test set.

All in all, this means that the training was done based on 1.795.964 flows which were retrieved from two small business networks. The mixed test set then contained 4.835.419 retrieved during 6 days from the same networks.

Now that the changes in the data set were described in detail, the natural question that arises is, how this influenced the detection rates of the classification algorithms. As it was done before the different algorithms were applied on sets of feature vectors retrieved within different time frames. The following table 6.9 provides an overview showing the detection rates of the same algorithms that have been used before. For the sake of comparable results all parameters of the different algorithms were kept the same.

Comparing these results to the previously retrieved results it is noticeable, that the detection has increased a little bit, as it was expected. This time the best result was achieved by the C.5 decision tree algorithm with a time frame size of 100 seconds. Hence the random forest and the SVM perform actually as well as the C4.5 algorithm, with less than half a per cent of difference.

Algorithm	60Sec	80sec	100sec	120sec
DecisionTree	94.13%	94.90%	95.10%	95.35%
RandomForest	94.45%	94.98%	95.03%	95.17%
SVM	92.7	94.07%	94.86%	95.10%
NaiveBayes	23.20%	23.44%	23.87%	24.31%
MLP	91.5%	89.59%	91.45%	90.46%

Table 6.9: Overview detection rates of different algorithms.

Choosing a time frame of 100 seconds still seems to be a good compromise, since the detection rates rarely increase above this level. All algorithms achieved their best detection rate at 120 seconds.

The figure 6.9 illustrates these detection performances

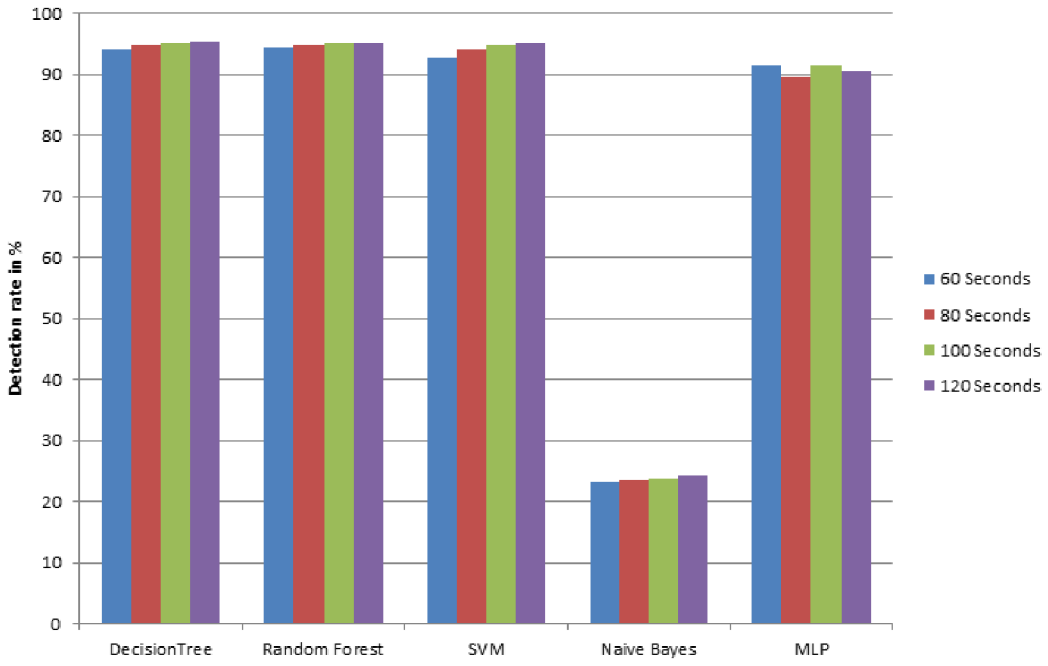


Figure 6.9: Comparison of detection rates 60 to 120 seconds.

Again it is visible from figure 6.9 that the Naive Bayes algorithm performs very badly in relation to the other classifiers. As mentioned before, this was expected and it was just included into this test, to prove that it is not the classifier of choice, because it relies on an assumption that does not hold true with this kind of data set. The Naive Bayes classifier assumes the independence of each variable in the feature vector, which will never be true in this data set. For example, the feature outgoing packets is also related to the number of SMTP connections or DNS requests. These results again empirically demonstrate that the theoretical hypothesis, that this classifier will not work, holds true.

Even though the Multilayer perceptron was proofed to be able to perform best on any data set that is classifiable in any way, it did not perform best in this experiment, but still had a detection rate similar to the other algorithms such as SVM or Random Forest. This is due to the previously mentioned constraint that any neural network has to be seen as a black box. Thus its internal learn model is hidden and not understandable for a human, therefore

optimizing the network training is an iterative task with slightly changed parameters (size of network, learning rate, amount of adaption in backpropagation, and evolutions for learning). Therefore it is obvious, that, even though the Multilayer Perceptron could theoretically be the best classifier, the actual detection rate reached might be limited due to practical reasons.

The presented results of the Multilayer Perceptron have been achieved on a network with 11 perceptrons on the input layer one hidden layer with 6 perceptrons and one on the output layer (binary decision). It is obvious that the number of perceptrons in the hidden layer has to be equal or lower than the number of perceptrons in the input layer. The learning rate was set to 0.08 and the adaptation factor of the weights during the backpropagation was set to 0.05. With these parameters, the network was trained over 3500 evolutions.

The following list provides an overview of the parameters used with the different algorithms that were presented above:

- DecisionTree: minimum number of instance per leaf = 2.
- RandomForest:
  - Number of trees to generate = 10.
  - No limititation to the depth.
- SVM:
  - Kernel size = 7.
  - Cache size = 128 MByte.
- NaiveBayes: no parameters available
- MLP:
  - 3 Layers: 11 perceptrons on input layer, 6 perceptrons on hidden layer, 1 perceptron on output layer.
  - Learning rate: 0.08.
  - Momentum: 0.05
  - Evolutions: 3500

All Experiments were carried out with Weka [[46]] in Version 3.7. Parameters that were not mentioned to be changed, have been kept on the default value provided by Weka.

Nevertheless, the previously mentioned imbalance in the number of positive feature vectors retrieved from the NATed network in relation to the negative feature vectors might have influnenced the models that were constructed by the different classifiers. Especially the Naive Bayes classifier is known to be strongly influenced by such an imbalance and in fact showed the worst detection rate of all the algorithms that were analyzed. Thus, it was also important to analyze the influence of this imbalance in the training set and the test set respectively.

The naturally arising question is, how to provide such balanced data sets, without manipulating the data sets in a way that learned model is not some sort of biased again. It is obvious that the samples have to be removed in the last step of the feature transformation process. This means that flows have to be aggregated into the feature vectors in the exactly the same way as before. Thus the balancing must be done on the feature vectors that are then used for training and testing.

It is important to mention that there is no way to perform this task in a way that completely ensures that the resulting data is not biased in any direction. Furthermore, it is obvious

that it is neither possible to do the selection manually, because of the huge number of samples nor would it make sense to do it in this way, because this highly increases the probability that the resulting balanced set of feature vectors is biased in some way. The only option to avoid getting biased data is by selecting features from the larger class of samples randomly. This means that the exactly the same amount of positive (NATed) feature vectors will be selected randomly, from the larger positive sample set, as the negative (non NATed) sample set contains.

Applying this method leads to the data set, shows how the imbalance from before influenced the model creation process and therefore also the detection performance in the end. The figure 6.10 illustrates the balanced feature extraction process for the Training Data set:

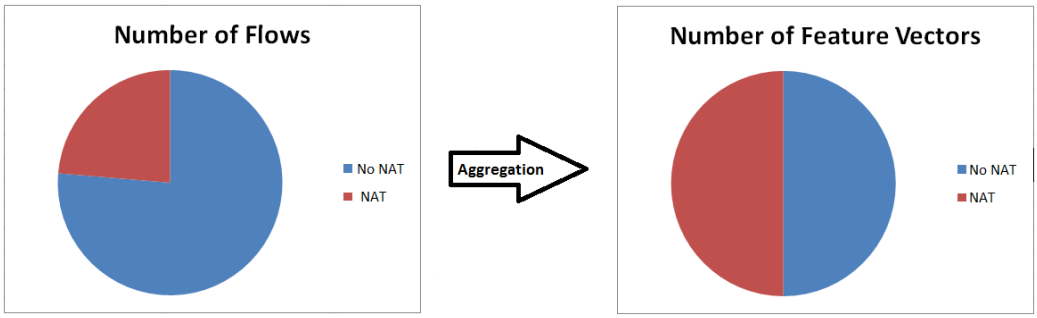


Figure 6.10: Proportions of balanced training feature vector set.

The data set for training contained 25.256 feature vectors in total, which were equally distributed now. Based on the same methodology as was used for the training set was of course also applied on the set of feature vectors used for testing. This resulting testing set contained 65608 feature vectors in total. The following figure illustrates the proportions of the used test set:

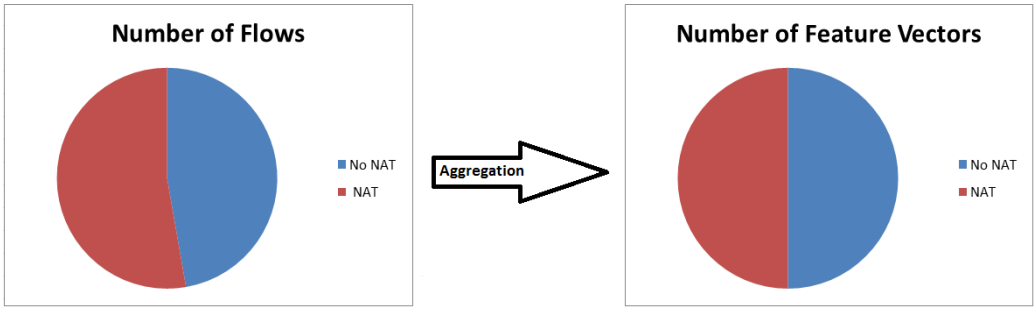


Figure 6.11: Proportions of balanced test set based on random selection.

After the equally balanced data sets for training and testing were created, the algorithms from before were applied again. It is expected that the Detection rates might drop a little for all the algorithms besides the Naive Bayes algorithm. This is because every model will also learn the probability distribution somehow, but the Naive Bayes algorithm is strongly affected by an imbalance in the data sets. Thus, this algorithm should show higher detection rates on the new data set. Since it was shown before that the data influence of the time frame is not very high and the best results in the last experiment could be achieved with a time frame for aggregation of 120 Seconds, the following analysis was just done with this time frame. The table [6.10] lists the detailed results that could be achieved on the new data set:

Algorithm (120 sec)	imbalanced	balanced
C4.5	95.35%	89.35%
RandomForest	95.17%	89.91%
SVM	95.10%	81.29%
NaiveBayes	24,31%	69.05%
MLP	90,46%	70,01%

Table 6.10: Overview Detection Rates of different Algorithms

It is important to keep in mind that the training data was again based on two days and the testing data again on 6 days. The only difference was, that within these two or respectively 6 days only a randomly selected subset was used.

An interesting observation that can be made from the table **??** above is that the detection rates indeed drop a few percent for most of the algorithms. In case of the Decision Tree algrithms this drop is rather small, especially with the Random Forest algorithm. The reason for this is probably that it was not strongly influenced by imbalance. This also proves its generalization performance. That means, it probably learned a more general model that was not based on the distribution of samples.

Furthermore, as expected, the Naive Bayes algorithm's detection rate increased dramatically. This clearly shows that the hypothesis regarding this classification algorithm from before was right. This classifier now classifies more than two out of three samples right, which brings it in a range where it becomes useful, but still it cannot compete with detection rates of Random Forest or the SVM which still have detection rates in the range of 90%. The figure 6.12 illustrates the differences in the detection rates between the unbalanced and the balanced data sets:

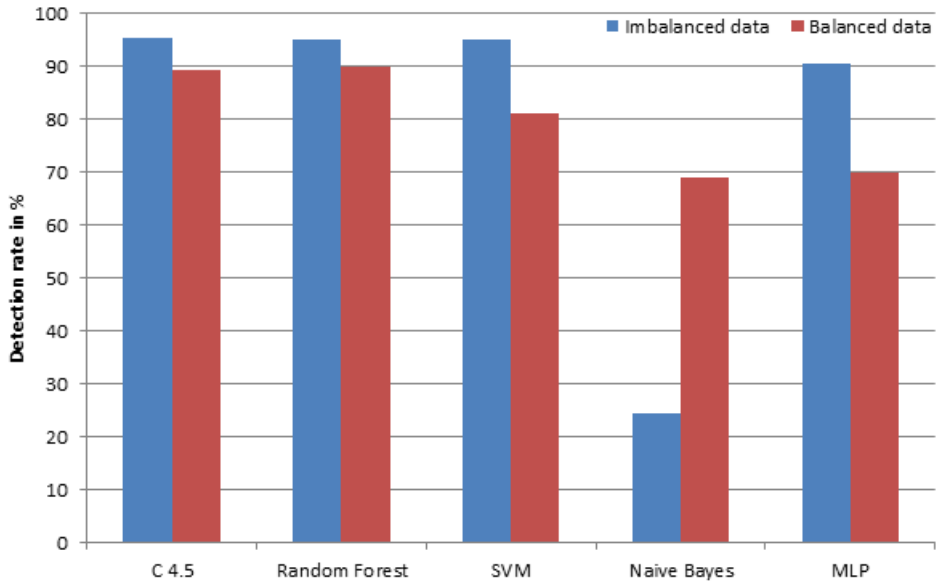


Figure 6.12: Comparison of the detection rates before and after balancing the data set.

In figure 6.12 it is clearly visible that the detection rate of the Multilayer Perceptron dropped down about 20%. This clearly shows that also this classifier learned the imbalance of

Rank	Feature	Info Gain
1	numOutBytes	0.208457
2	numUDP	0.11267
3	numTCP	0.093539
4	numOutPackets	0.060406
5	numInBytes	0.059318
6	numInPackets	0.052855
7	numDNSReq	0.006267
8	numMailCon	0.005503
9	numSMTP	0.00079
10	numSYN	0.000532
11	numRst	0.0

Table 6.11: Information gain of features.

the data set. Hence, as discussed before, the Multilayer perceptron in theory can be optimized to achieve the highest detection rates for a certain classification problem. In practice this often means that the parameters of it have to be adjusted iteratively. For example, increasing the number of hidden layers might be one of these parameters that could be changed.

Feature Ranking

Based on a feature vector set, retrieved with 100 second aggregation, an analysis on the relevance of each feature was carried out. For this analysis, each of the previously described features was ranked by its information gain ratio, which is calculated based on the entropy each feature contains in the data set. The outcome of this analysis is presented as a ranking in table 6.11.

The feature ranking was done with InfoGainAttributeEval algorithm in Weka.

It is clearly visible that the number of outgoing bytes attribute in the feature vector contains the most information and is therefore ranked highest. Due to the fact that the data sets did not contain flows with the Reset flag set it i ranked lowest.

As previously described, this was more or less expected since modern NAT-gateways (routers) normally do not send these flags, in case unintended connection attempts reaching the outside interface of the NAT-gateway. This is often done to reduce the information sent back to an attacker for example in case of a portscan. Nevertheless, it might occur more often on legacy or private equipment.
Therefore it is left in the feature set.

6.4 Summary

In this chapter the developed classifiers were presented and analyzed.

The first classifier presented in this chapter was based on special flows that are called SYN-flows within this thesis. For this classifier, a technique from the field of passive OS fingerprinting was successfully adapted to the NetFlow data set.

The next classifier presented was based on special patterns that are introduced by the NAT-gateways themselves. The classifier is based on observing the used source ports for new connections over a certain fixed time frame. In case multiple hosts try to use the same port numbers for connections to the public internet the NAT-gateway has to solve this conflict. It could be recognized that this often results in unique patterns, since many NAT-gateways simply increase the port numbers to solve this conflict.

The third approach presented in this chapter was based on analysis of the user behaviour in certain fixed time frames. This approach was based on machine learning algorithms.

After the detailed presentation of the classifiers, they were tested on different data sets. All of the classifiers were first tested on smaller simulated data sets. These data sets were sometimes to small and did not contain all the patterns for all classifiers. For example, the simulated environment did not contain SYN-flows since these flows normally show up, if some kind of abnormal traffic is created. Such abnormal traffic might be for example a slowly or not responding Mail-server. It turned out that both, the SYN-flow based approach and the Source Port related classifier, work best under these abnormal conditions.

In contrast to this, the user behaviour based approach showed very good results on both the local network setup and on the huge data set for the business use case. With this approach repeated detection rates of around 90% were achieved.

CHAPTER 7

Conclusion

This chapter first provides a summary and a short discussion of the results that were achieved in this thesis. Subsequently, a decision fusion concept is presented that has been developed during this thesis, but could not be applied in practice anymore. Finally other open research topics are presented.

7.1 Summary and Discussion

In this thesis it was shown that NAT can be detected based on NetFlow data. This is in contrast to most other NAT detection approaches that were presented before. The approach of detecting NAT based on the analysis of the user behaviour, that was introduced by others before, was developed further and it was shown that very high detection rates can be achieved with this approach. It is also important to point out, that for the user behaviour based approach a detailed analysis on selecting the best machine learning algorithm was carried out. It turned out that the decision tree approaches achieved the best results, but other approaches such as support vector machines and multilayer perceptrons are capable to achieve results in an equal range.

In addition to the user behaviour based detection approaches two further approaches were developed. These were applied on different levels of the communication.

The first one of these two approaches was based on a pattern that was introduced by many NAT gateways themselves. This detection approach is based on detecting sequences in the used source ports. It turned out during the analysis that these patterns occur, if multiple hosts on a local network try to use the same source ports for their communication with the same resource on the public internet. In this thesis, a detection algorithm for this pattern was developed and applied tested on flow data sets. The tests showed that this classifier works best in case of high load even, if this high load is caused by abnormal or malicious traffic.

For the last detection approach, a technique from the field of passive remote OS fingerprinting were successfully adapted in a way to work on NetFlow data sets as well. This method is based on special flows that solely consist of SYN packets that were aggregated into one flow. It has been shown that these flows can be filtered out by the pattern "....S." in their TCP flag attribute. The basis for this detection approach is the knowledge that the SYN packets of different operating systems normally have a unique size, which can then be used to determine the operating system the packets originate from. The method presented in this thesis had to overcome the problem that NetFlow only contained high-level aggregated statistical information about connections and not the packets themselves. The solution presented in this thesis is based on the three steps "filtering for SYN flows", "Computing the average size of the packets of a SYN flow" and finally "Checking this size against a list of known sizes". As with the previous approach also this one turned out to work best on abnormal and malicious traffic like portscans from the inside to some outside destination or unavailable SMTP servers.

For testing and proving the concepts, the algorithms were applied to different data sets. All algorithms have first been applied on a flow data retrieved from a simulation environment, which was also created within this thesis. After this first analysis, all the detection algorithms also were applied on a huge data set retrieved by a German ISP from two of their business customers networks. Since the algorithms were applied on a real world scenario, they can be considered to be very reliable.

Three different detection approaches were developed and tested. It made sense to develop a concept for fusing the different approaches into one system. The aim of such a system would be to combine the advantages of all approaches in one system. In the scope of this thesis this concept could not be applied in practice any more. Therefore, a detailed description of it is included in the future work section later in this final chapter.

It is also important to point out that the presented detection approaches also support the remote size estimation of a network. This is for example of relevance for the use case botnet size estimation. The contribution of the NAT detection to this problem is, that a detected NAT can be used as the lower bound for the further estimation, since it is known that at least two hosts are active and sharing the same internet line. Especially the SYN flow based approach is also able to reliably determine network sizes up to a size of 4 hosts, if at least 4 different operating systems are used.

The following section lists the main contributions of this thesis.

7.2 Contributions

Since this thesis combines methods from a variety of different research fields, this section aims to provide a compact outline of the main contributions of the thesis. These contributions are:

- **Simulation environment. This was developed for testing and emulating local residential networks as well as business equipment.**

- **Browser testing framework. It was shown that browser testing framework can support the simulation of real user behaviour.**

- **Sequences in source ports. The analysis showed that NAT gateways often introduce patterns, by which they can be detected.**

- **Average SYN packet size. A technique from passive OS fingerprinting was successfully adapted to NetFlow data and used for NAT detection.**

- **User behaviour. A new feature set based on user behaviour observation was developed and tested with great success, by means of high detection rates that could be achieved by different algorithms based on this feature set**

- **Determine lower bound of network size. This thesis also contributes to estimating the size of the network behind NAT. This is due to the fact that a detected NAT means that a network at least consists of two hosts. In addition the SYN flow classifier can determine the lower bound of a network size up to the number of 4 hosts, in case 4 different operating systems are used.**

- **Decision fusion. Finally a new ensemble NAT detection concept was proposed. This concept aims to fuse the decisions of the different classifiers by additionally using some sort of anomaly detection.**

In addition to the contributions listed above, this thesis also contributes to the more general question of, how much information is really contained in NetFlow data. This might also be an interesting aspect in the discussion about data preservation which is normally based on the same or very similar data set.

7.3 Future Work

This section is split in to two subsections. First the previously mentioned decision fusion concept is presented in more detail. Subsequently, a short outline of other still open research questions is provided.

One major approach regarding the detection approaches presented within this thesis might be to fuse the different detection approaches into one system to combine their individual advantages.

7.3.1 Ensemble NAT detection

As described before, three different approaches were developed and tested within this thesis. In addition, it was shown that the detection approaches had different advantages and disadvantages. To combine the advantages of each of the classifiers, a fusion concept has already been developed, but could not be tested in the scope of this thesis.

The figure 7.1 shows the components of the described ensemble system integrated into the previously described general component and data flow model.

Previously, the different detection approaches were described in detail. One aspect of these details were the different operational boundaries for each of the three feature sets (SYN, additive Src-Port and behavioural). In addition, it is important to remember that each of these feature sets has besides these operational boundaries further individual advantages or disadvantages.

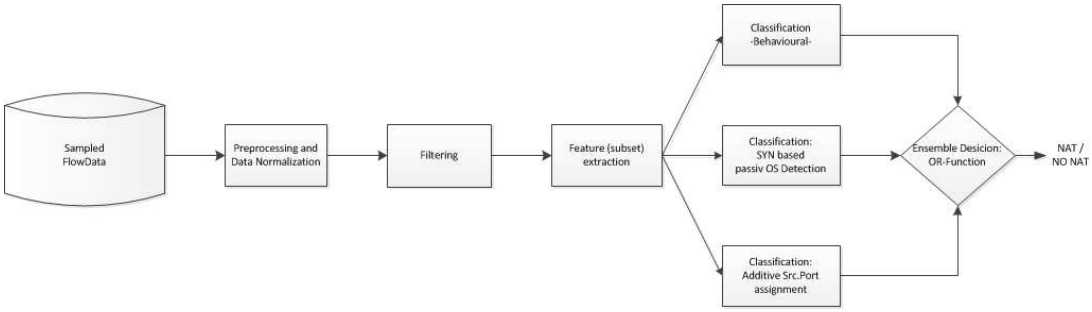


Figure 7.1: Ensemble NAT detection system.

The goal of the NAT detection system is to combine the advantages of each of these classification approaches. This could be done by using some sort of ensemble method. For the detection of NAT the overall goal is to have a binary decision in the end that states whether flows from an IP address come from a NAT or directly from a single host.

In general, the decision fusion function can be seen as a summation of the single decisions of each of the three classification approaches. In such a scenario each of decisions of the three detection approaches would be considered with a weight that depends on its measured detection rate. Therefore for a naive first approach this function might be denoted as follows:

$$D_f = \frac{1}{n} \sum_{i=1}^{n} w_i \times d_i \tag{7.1}$$

Where D_f is the fused decision score. The number of decision components is denoted by n. Factor w_i represents the weight of each decision from the decision components and d_i is the decision of each of the decision components. To retrieve a rational value between 0 and 1 it is also necessary to divide the decision by the number of decision components.

In this scenario it is of course assumed that each component also provides a decision in the range of 0 to 1 that might be either binary or a rational number. In a naive case a value of D_f which is above 0,5 might be seen as true, hence this threshold of course depends on the actual system environment where it is applied on.

In such a basic representation the decision of the components would be independent of the type of traffic. This means any abnormal traffic would not be part of the decision fusion process. Thus the above formula has to be reformulated to represent also the input of an external network intrusion detection system or anomaly detection system in general. This decision fusion score could for example be computed in the following way:

$$D_f = (1 - a) \times \left(\frac{1}{n} \sum_{i=1}^{n} w_i \times d_i \right) + a \times \left(\frac{1}{m} \sum_{j=1}^{m} w_j \times d_j \right) \tag{7.2}$$

Where D_f again is the total decision fusion score. j denotes the number of classification components for abnormal traffic and the factor a is the degree of abnormality determined by an external anomaly detection component, which will be a rational number in the range between 0 and 1. The summand on the left side denoted as before and representing the classification

component(s) for normal traffic (e.g. user behaviour based classification), it is included in the overall decision based on the factor $(1 - a)$. This factor decreases, the higher the value a becomes or expressed in other words the more abnormal the traffic, the lower this factor will be.

Any classification component known to operate best on abnormal traffic (e.g. SYN flow based detection) is represented by the right hand side of the term. The decisions made by these will be stronger represented in the overall decision fusion score, the higher the factor a becomes. In other words, with increasing abnormality of the network traffic, the decisions of these classifiers will be respected much more.

In the previously mentioned principle it was assumed that the detection rates are used for the factors w for weighting the decisions of each approach.

These have to be determined by an expert for a certain system the classifiers should be applied on. The natural question might arise at which score the decision, whether NAT was fund or not, is made true or false. In other words a threshold has to be defined for D_f. As simple naive approach this might just be again set to 0,5.

Bearing in mind that this decision is based on weighted summation and a threshold a self-evident idea is to use a neural network or especially a perceptron based approach, since this perfectly fits to the definition of a perceptron, which is also based on weighted summations and a threshold as an output function. One might then train the decision system instead of determining the threshold value by oneself.

The figure 7.2 illustrates this concept:

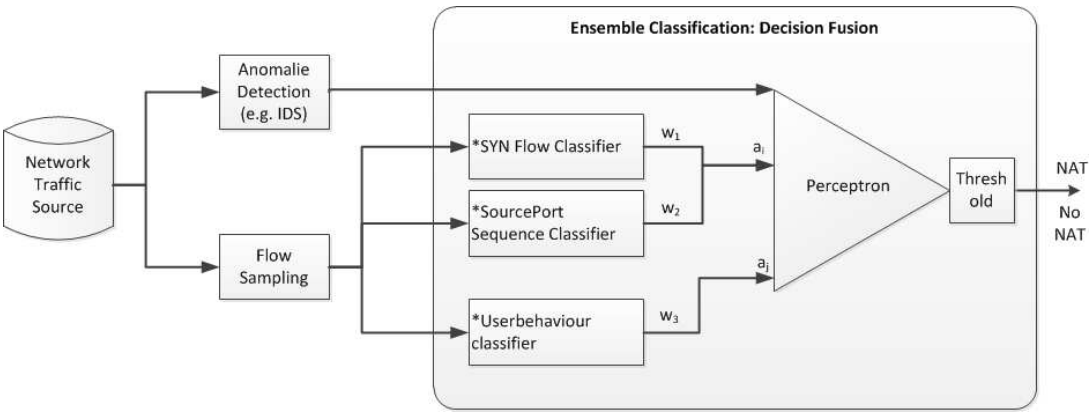


Figure 7.2: Perceptron based Decision Fusion Score.

7.3.2 Further open research questions

In addition to implementing and testing the previously mentioned decision fusion concept, there are other open research questions in this field, which need to be answered in the future.

For example, it was also mentioned in the thesis that NAT detection could probably also be done based on detecting multiple active instances of certain applications such as Skype or DropBox accounts. These accounts often run in the background and regularly keep their connection alive or check for updated data in the cloud. A variety of similar applications such as GoogleTalk, Skydrive, GoogleDrive or music streaming clients such as Spotify. All these applications have in common that they only allow one running instance per user, per host. If one can detect more running instances of one of these applications remotely on one IP address, this

means that different users are active, which normally use different hosts/devices. Detecting applications based on their footprint in network traffic is not trivial and becomes even more difficult if applied on NetFlow data instead of the single network packets.

In addition, the precise size estimation is still an open topic. Even though this thesis contributes to this topic, this contribution is limited to setting the lower bound of a networks size. Estimating the size of large networks is not possible with this method. For example, it is still not possible to distinguish a network containing 20 hosts from one containing 30 hosts. This task is relatively challenging since the user behaviour differs from user to user and also the user topology differs from network to network. The main problem to solve in this domain is the generalization of the data sets and models.

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *SIGMOD Rec.* 22.2 (June 1993), pp. 207–216.

[2] Avira Antivir. URL: http://www.avira.com/de/index(lastvisited:14.02.2013).

[3] Steven M. Bellovin. "A technique for counting natted hosts". In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. IMW '02. Marseille, France: ACM, 2002, pp. 267–272.

[4] Avrim L. Blum and Pat Langley. "Selection of relevant features and examples in machine learning". In: *Artificial Intelligence* 97 (1997), pp. 245 –271. URL: http://www.sciencedirect.com/science/article/pii/S0004370297000635.

[5] David G.Stork Chuck Lam. "The Handbook of Brain Theory and Neural Networks". In: ed. by Michael A. Arbib. second edition. MIT Press, Cambridge, 2001. Chap. Learning network topology.

[6] Cisco. *NetFlow Collection*. Table taken from. URL: http://www.cisco.com/en/US/docs/net_mgmt/netflow_collection_engine/3.5/user/guide/format.html.

[7] Cisco. *NetFlow Reliable Export With SCTP*. URL: http://www.cisco.com/-en/US/-docs/ios-xml/ios/-netflow/configuration/-12-4t/nflow-export-sctp.html\#GUID-EEE2123A-8FE1-4AB8-A6F0-42255B6C98B6.

[8] Cisco. *Overview of Cisco NetFlow versions:* URL: http://netflow.caligare.com/netflow_v5.htm.

[9] NetFlow Export Cisco. Last visited 01.11.2012. URL: http://www.cisco.com/-en/US/-docs/ios-xml/ios/-netflow/configuration/-12-4t/nflow-export-sctp.html\#GUID-EEE2123A-8FE1-4AB8-A6F0-42255B6C98B6.

[10] *Dynamips*. URL: http://www.gns3.net/dynamips/,(Lastvisited:17.02.2013).

[11] Tristan Fletcher. "Support vector machines explained". In: *Tutorial paper., Mar* (2009).

[12] Cisco Flow-Collector. Cisco; NetFlow Collection. 2013. URL: http://www.cisco.com/en/US/docs/net_mgmt/netflow_collection_engine/3.5/user/guide/overview.html,(Lastvisited:15.02.2013).

[13] ForeScout. URL: http://www.forescout.com/wp-content/media/ForeScout-Tech-Brief-NATDetection-110111-FINAL1.pdf,(Lastvisited01.10.2012) ..

[14] Johannes Fürnkranz. "A study using n-gram features for text categorization". In: *Austrian Research Institute for Artifical Intelligence* 3.1998 (1998), pp. 1–10.

[15] *GNS3*. URL: http://www.gns3.net/,(Lastvisited:17.02.2013).

[16] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1157–1182.

[17] *KDD Cup*. URL: http://kdd.ics.uci.edu/databases/kddcup99/task.html(lastvisited:18.02.2013).

[18] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. "Remote Physical Device Fingerprinting". In: *IEEE Trans. Dependable Secur. Comput.* 2.2 (Apr. 2005), pp. 93–108.

[19] Vojtěch Krmíček, Jan Vykopal, and Radek Krejčí. "Netflow Based System for NAT Detection". In: *Co-Next Student Workshop'09: Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*. 2009, pp. 23–24.

[20] Rui Li et al. "Passive NATted Hosts Detect Algorithm Based on Directed Acyclic Graph Support Vector Machine". In: *Proceedings of the 2009 International Conference on Multimedia Information Networking and Security - Volume 02*. MINES '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 474–477.

[21] WarrenS. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". English. In: *The bulletin of mathematical biophysics* 5 (4 1943), pp. 115–133.

[22] *Mozilla Thunderbird*. URL: http://www.mozilla.org/de/thunderbird/(lastvisited:20.01.2013).

[23] Kevin P. Murphy. 2006. URL: http://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/reading/NB.pdf,(Lastvsited:16.02.2013.).

[24] Michael Negnevitsky. *Artificial Intelligence - A Guide to intelligent Systems*. Third. Pearson, 2011.

[25] Hai Thanh Nguyen. "Reliable Machine Learning Algorithms for Intrusion Detection Systems: Machine Learning for Information Security and Digital Forensics." PhD thesis. Gjovik University College, 2012.

[26] *Parallels Desktop*. URL: http://www.parallels.com/de/(lastvisited:15.02.2013).

[27] P. Phaal. *"Detecting NAT Devices using sFlow"*. URL: http://www.sflow.org/detectNAT/.

[28] Min Qin and Kai Hwang. "2004, Frequent Episode Rules for Intrusive Anomaly Detection with". In: *Internet Datamining", USENIX Security Symposium, submitted Jan.27*. 2004.

[29] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[30] John Ross Quinlan. *C4. 5: programs for machine learning*. Vol. 1. Morgan kaufmann, 1993.

[31] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[32] Li Rui et al. "Remote NAT Detect Algorithm Based on Support Vector Machine". In: *Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on*. 2009, pp. 1 –4.

[33] K. Koutroumbas S. Theodoridis. *Pattern Recognition*. Third. ELEVISTER (USA), 2006.

[34] K. Koutroumbas S. Theodoridis. *Pattern Recognition*. Fourth. ELEVISTER, 2009.

[35] *Selenium*. URL: http://docs.seleniumhq.org/,(Lastvisited:18.02.2013).

[36] Sebastian Seung. 2002. URL: http://hebb.mit.edu/courses/9.641/2002/lectures/lecture04.pdf,(Lastvisited13.12.2012).

[37] sFlow: URL: http://www.sflow.org/,Lastvisited:15.02.2013.

[38] A. SuÃ¡rez SÃ¡nchez et al. "Application of an SVM-based regression model to the air quality study at local scale in the AvilÃ©s urban area (Spain)". In: *Mathematical and Computer Modelling* 54.5-6 (2011), pp. 1453 –1466. URL: http://www.sciencedirect.com/science/article/pii/S0895717711002196.

[39] *softflowd*. URL: http://code.google.com/p/softflowd/,(Lastvisited:17.02.2013).

[40] *Source of Figure*. URL: http://www.lancope.com/blog/netflow-v5-vs-netflow-v9/.

[41] *Source of Figure*. URL: http://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html.

[42] Brett Stone-Gross et al. "Your botnet is my botnet: analysis of a botnet takeover". In: *Proceedings of the 16th ACM conference on Computer and communications security*. ACM. 2009, pp. 635–647.

[43] *tcpdump*. URL: http://www.tcpdump.org/,(Lastvisited:17.02.2013).

[44] Vladimir N Vapnik. "An overview of statistical learning theory". In: *Neural Networks, IEEE Transactions on* 10.5 (1999), pp. 988–999.

[45] *"VirtualBox "*. URL: https://www.virtualbox.org/(lastaccessed:16.02.2013).

[46] Weka: URL: http://www.cs.waikato.ac.nz/ml/weka/,(Lastvisited10.12.2012).

[47] Forensics Wiki. URL: http://www.forensicswiki.org/wiki/NAT_detection, (Lastvisited01.10.2012)..

[48] Sean Wilkins. *Basic NAT Concepts and Configuration*. Cisco Press. 2011. URL: http://www.ciscopress.com/articles/article.asp?p=1725268.

[49] *Wireshark*. URL: http://www.wireshark.org/(lastvisited:17.02.2013).

[50] *WMware*. URL: http://www.vmware.com/(lastaccessed:15.02.2013).