BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

BEHAVIOUR-BASED IDENTIFICATION OF NETWORK DEVICES

IDENTIFIKACE ZAŘÍZENÍ NA ZÁKLADĚ JEJICH CHOVÁNÍ V SÍTI

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MICHAEL ADAM POLÁK**
AUTOR PRÁCE

SUPERVISOR **Ing. LIBOR POLČÁK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

Department of Information Systems (DIFS) Academic year 2019/2020

Master's Thesis Specification

22644

Student: **Polák Michael Adam, Bc.**

Programme: Information Technology Field of study: Information Technology Security

Title: **Behaviour-Based Identification of Network Devices**

Category: Networking

Assignment:

1. Study methods of device identification based on identifiers carried inside network traffic (such as IP address, MAC address, Cookies, or hidden identifiers such as HTTP fingerprinting and clock-skew).
2. Study approaches used to identify or classify devices based on communication metadata (eg. based on IPFIX, NetFlow, or DNS queries).
3. In consultation with Cisco Systems, s.r.o. and the supervisor propose an algorithm for network device identification.
4. Implement the proposed method.
5. Perform experiments evaluating the implemented method.
6. Evaluate the work and suggest possible improvements.

Recommended literature:

- Polčák, Libor. *Lawful Interception: Identity Detection*. Brno, 2017. Ph.D. thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Miroslav Švéda.
- Kumpošt, Marek. *Context Information and UserProfiling*. Brno, 2009. Ph.D. thesis. Masaryk University. Supervisor Václav Matyáš.
- Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine: *Browser Fingerprinting: A survey*. ArXiv e-prints, e-print 1905.01051, 2019.

Requirements for the semestral defence:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Polčák Libor, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	June 3, 2020
Approval date:	October 29, 2019

Abstract

This thesis deals with the topic of identifying devices based on their behaviour. With the increasing number of devices on the network, it is becoming more and more important to be able to identify these devices based on their behaviour, due to the increased security risks. General networking concepts and multiple methods that have been used in the past to identify devices are discussed throughout the work. Subsequently, machine learning algorithms and their advantages and disadvantages are introduced. Finally, this thesis tests two traditional machine learning algorithms and proposes two new approaches to network device identification. The resulting final algorithm achieves the accuracy of 89% on a real life data-set with over 10,000 devices using a set of only eight features.

Abstrakt

Táto práca sa zaoberá problematikou identifikácie sieťových zariadení na základe ich chovania v sieti. S neustále sa zvyšujúcim počtom zariadení na sieti je neustále dôležitejšia schopnosť identifikovať zariadenia z bezpečnostných dôvodov. Táto práca ďalej pojednáva o základoch počítačových sietí a metódach, ktoré boli využívané v minulosti na identifikáciu sieťových zariadení. Následne sú popísané algoritmy využívané v strojovom učení a taktiež sú popísané ich výhody i nevýhody. Nakoniec, táto práca otestuje dva tradičné algorithmy strojového učenia a navrhuje dva nové prístupy na identifikáciu sieťových zariadení. Výsledný navrhovaný algoritmus v tejto práci dosahuje 89% presnosť identifikácii sieťových zariadení na reálnej dátovej sade s viac ako 10000 zariadeniami.

Keywords

Machine learning, behaviour based identification, network device behaviour, user profiles, classification, decision tree, Naive Bayes Classifier, computer networks, security, device tracking, device identification, text similarity, outlier detection

Kľúčové slová

Strojové učenie, identifikácia na základe správania, správanie sa sieťových zariadení, profily používateľov, klasifikácia, rozhodovací strom, naivný bayesovský klasifikátor, počítačové siete, bezpečnosť, sledovanie zariadení, identifikácia zariadení, podobnosť textu, detekcia odchýliek

Reference

POLÁK, Michael Adam. *Behaviour-Based Identification of Network Devices.* Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

Rozšírený abstrakt

S rastúcim počtom zariadení pripojených k počítačovej sieti je neustále dôležitejšie pre sieťových administrátorov schopnosť identifikovať zariadenia na ich sieťach. Dôvodom na identifikáciu zariadení je zvyšujúci sa počet útokov na počítačové siete, ktorý rastie každým dňom [17]. Z toho dôvodu je potrebné vyvíjať metódy, ktoré identifikujú sieťové zariadenia len na základe ich správania sa v sieti. Na základe týchto algoritmov je potom možné porovnávať aktuálne správanie sa zariadenia oproti vzorom v minulosti a následne rozhodnúť, či daný model správanie korešponduje so zariadením, alebo je zariadenie podvrhnuté a je potrebné túto komunikáciu ukončiť.

Cieľom tejto práce je navrhnúť algoritmus za použitia algoritmov strojového učenia, ktoré jednoznačne identifikujú sieťové zariadenia len na základe ich správania sa v sieti, bez znalosti jednoznačných identifikátorov ako je MAC adresa. Následne je potrebné, aby navrhovaný algoritmus neustále udržiaval uchovávané modely zariadení aktuálne, bez potreby veľkých pamäťových požiadaviek. Poslednou požiadavkou je schopnosť identifikácie nových zariadení a potreba minimálneho pretvárania modelu alebo modelov pri pridávaní nových zariadení do siete. Táto práca otestuje dva tradičné algoritmy strojového učenia ako sú Naivný Bayesovský Klasifikátor a Rozhodovacie Stromy. Následne sú navrhnuté dva nové algoritmy inšpirované algoritmami bežne používanými na kontrolu podobnosti textov. Tieto algoritmy sú nakoniec najúspešnejšími metódami na identifikáciu a sledovanie sieťových zariadení.

Prvým z algoritmov inšpirovaným podobnosťou textov je metóda s názvom Nearest Neighbor Combined Model. Táto metóda počas trénovacej fáze algoritmu agreguje toky na základe ich MAC adresy a používateľského mena. Následne je na tieto agregované dáta aplikovaná frekvenčná analýza, ktorej výstupom je vysokodimenzionálny vektor frekvencii jednotlivých elementov. Tieto vektory sú nazývané profilmi zariadení. Po uplynutí trénovacej fáze, ktorá trvá jednu hodinu následuje klasifikačná fáza. V tejto fáze sú profily agregované v 5 minútových intervaloch na základe zdrojovej IP adresy a používateľského mena. Táto kombinácia reflektuje neznalosť MAC adresy a zároveň je dostatočná na rozlíšenie zariadení od seba navzájom, avšak nie je dostatočná na jednoznačnú identifikáciu daného zariadenia, keďže zdrojová IP adresa sa môže s časom zmeniť a používateľ môže vlastniť viac zariadení. Nakoniec je používaná kosínusová podobnosť na nájdenie najbližšieho "veľkého" profilu k aktuálne klasifikovanému profilu zariadenia. Po priradení profilu k zariadeniu sú tieto dáta pridané k "veľkému" profilu na spresnenie predpovedí v budúcnosti. Nevýhodou tejto metódy je však jej neschopnosť identifikovať nové zariadenia a zároveň neustále narastajúca veľkosť profilu, ktorý by sa časom nezmestil do pamäti RAM, hlavne pri sledovaní zariadení na univerzitách alebo vo veľkých firmách. Z toho dôvodu bola navrhnutá druhá metóda, ktorá rieši nedostatky aktuálne popísanej metódy.

Druhou navrhovanou metódou na identifikáciu a sledovanie zariadí sieti je metóda s názvom k-NN with Segmented Profiles Model. Tento model je inšpirovaný algoritmom popísaným v predošlom odseku. Jeho hlavným rozdielom je, že unifikuje dĺžku časového okna počas trénovania aj počas klasifikácie. Podobne ako predošlý algoritmus agreguje toky z NetFlow na základe zdrojovej IP adresy a používateľského mena, ale už v trénovacej fáze. Trénovacia fáza trvá tiež hodinu a v pamäti sa uchováva len posledných 12 profilov daného zariadenia. Následne je aplikovaná frekvenčná analýza na jednotlivé profily. Niektoré stĺpce sú však analyzované zvlášť ako napríklad zdrojová a cieľová IP adresa. Tieto stĺpce sú analyzované oddelene z dôvodu, že by mohli mať rovnaké hodnoty, avšak ich význam je úplne odlišný a bežnou frekvenčnou analýzou by sa stratil ich sémantický význam. Tento jav môže nastať v prípade, že dve zariadenia komunikujú medzi sebou a ich lease IP adresy by vypršal v rovnakom čase. Následne je možné, že DHCP server priradí IP adresy v opačnom poradí ako v predošlom prípade a teda by sa stratila smerovosť následnej komunikácie. Po už popísaných krokoch nasleduje fáza klasifikácie. V tejto fáze sú rovnakým spôsobom agregované toky a je aplikovaná aj frekvenčná analýza ako v trénovacej fáze. Avšak pre klasikovanie daného zariadenia je použitý algoritmus *k* Najbližších susedov. Po klasifikácii je nahradený najstarší z profilov identifikovaného zariadenia v prípade dvanástich profilov uchovaných v pamäti. V opačnom prípade je najnovší profil len pridaný k danému zariadeniu. Týmto spôsobom sa profily zariadení neustále obnovujú avšak pamäťová náročnosť s časom až tak radikálne nerastie ako pri predošlom algoritme. Profily zariadení sú uchovávané dočasne a po určitej dobe neaktivity sú zmazané, pre zamedzenie skladovania už neaktívnych zariadení v danej sieti. Tento algoritmus tiež implementuje detekciu odľahlých bodov pomocou k-NN algoritmu a z-skóre. Tieto metódy umožňujú navrhovanému algoritmu detekovať nové zariadenia. Jednotlivé modifikácie tohto algoritmu sú postupne testované a pomocou dosiahnutých výsledkov boli navrhované nové modikácie.

Posledný navrhovaný algoritmus dosiahol priemernú presnosť predikcie 89% počas testovaných 8 hodín z exportovaných dát poskytnutými spoločnosťou Cisco systems obsahujúcou cez 10000 zariadení. Nakoniec je táto metóda porovnaná s už existujúcimi metódami na jednoznačnú identifikáciu zariadení a sú porovnané ich výhody i nevýhody. Výsledky tejto práce môžu byť využité sieťovými administrátormi na automatizovanú identifikáciu sieťových zariadení na základe ich správania bez znalosti jednoznačných identifikátorov ako je MAC adresa.

Behaviour-Based Identification of Network Devices

Declaration

I, Michael Adam Polák, hereby declare that this thesis is entirely my work prepared under the supervision of Ing. Libor Polčák, Ph.D. All relevant sources of information are duly cited and included in the list of references.

. . . .
Michael Adam Polák
June 1, 2020

Acknowledgements

I would like to sincerely thank my supervisor Ing. Libor Polčák, Ph.D. for his valuable advice and guidance. Furthermore, I would like to thank Mgr. Jan Kohout for his advice and patience during the development of the proposed algorithms. Lastly, I would like to give my sincere thanks my parents for their continuous support during my studies.

Contents

Chapter 1

Introduction

With the constantly increasing number of devices that are connected to the network, it is becoming more and more important for network administrators to be able to identify devices on the network. The root of this necessity lies in the increasing number of attacks that are performed per day [17]. Therefore, methods that identify devices based on their behaviour are being developed to identify and verify whether the behaviour of a device is similar to the already existent user profile, or the device is malicious, and the communication needs to be blocked. This thesis tests two traditional machine learning algorithms and proposes two new approaches to network device identification and tracking. The methods described in this thesis can be used by network administrators for automated identification of devices on their network in contexts where unique identifiers such as the MAC addresses are not available.

In the past methods such as HTTP or TLS fingerprinting have been used to reliably identify devices [2, 13]. However, with the increase in network traffic loads and significant increase in database sizes of these fingerprint databases, new approaches to identify devices need to be explored [2]. Furthermore, research that has been done up to this point explores only methods that measure similarity with previously seen behaviour, as described in the thesis of Kumpost [20]. This approach, however, has high spatial and time demands over large data-sets, which renders these methods impractical in a production environment. Another example of a unique device identification method proposed by Kohno et al. is by using the clock-skew measurement [19]. However, in some cases the use of clock-skew measurement requires synchronized sampling, which is an active method of finding the unique identifier. The goal of this thesis is to develop a method that passively tracks devices. Therefore, this work explores new approaches with better performance using artificial intelligence.

This thesis describes networking fundamentals and methods that have been used to identify devices in the past such as clock-skew, TLS and HTTP fingerprinting in Chapter 2. The advantages along with disadvantages of the aforementioned methods are also discussed in this chapter. The following Chapter 3 introduces concepts that are used in machine learning that are suitable for device identification. Methods include the Decision Tree Classifier, Naive Bayes Classifier, k-Nearest Neighbors, and Term Frequency and Inverse Document Frequency that is often used in text analysis. Lastly, this chapter introduces methods used for outlier detection, which in the case of this thesis are used for new device detection.

The next Chapter 4 contains a detailed description of the two newly proposed algorithms for device tracking. Both of these algorithms are inspired by methods for text analysis.

The first proposed algorithm, described in the Section 4.1, analyzes and creates large device profiles where the closest neighbor is found using the cosine similarity distance metric. While this algorithm performs well, it has multiple downfalls, such as low precision and recall, which are caused by incorrect classifications. Another problem is that this algorithm is not capable of detecting new previously unseen devices that have appeared in the data-set after the training phase has been completed. Therefore, the following Section 4.2 describes a modification of the aforementioned algorithm that overcomes its deficiencies. The modification consists of the segmentation of the large profile into multiple smaller ones and introduces the *k-Nearest Neighbors* algorithm with cosine similarity as the distance metric. Further modifications have been done to the algorithm including methods of updating device profiles, which significantly improve the memory efficiency. Lastly, methods used for outlier detection such as the z-score and k-NN outlier detection are introduced to the algorithm to improve the detection of new devices. This final algorithm achieves the best performance from all of the tested methods in this thesis.

The final product of this thesis is an application that enables the user to easily create experiments and vary the parameters of the aforementioned methods. The tool is highly configurable and modular, which enables easy additions of new device identification methods in the future. The tool description, design and used technologies are described in the Chapter 5.

Afterwards, in Chapter 6, methods described in this thesis are tested on two real life data-sets provided by Cisco Systems. These data-sets provide a wide variety of user behaviours and availability of features, which thoroughly test the already existing and the two newly proposed algorithms. The parameters of the algorithm with the best performance based on the experiments performed in this chapter are summarized in the Appendix B. Lastly, this chapter compares the achieved results with already existing algorithms and points out their advantages and disadvantages.

Chapter 7 is the conclusion to this thesis and proposes further improvements to this work.

Chapter 2

Network Device Identification

This chapter serves as an introduction to networking concepts and user behaviour patterns that are going to be used as a base for behaviour based network device identification in this thesis. The beginning of this chapter describes networking fundamentals in Section 2.1, and attributes provided by protocols such as HTTP or TLS. The second half of this chapter examines methods that have been used for device identification in a similar context, such as clock-skew measurements or user agent re-identification. Furthermore, techniques regarding fingerprint creation from the HTTP and TLS protocols are discussed in Sections 2.3 and 2.4 respectively. Lastly, the NetFlow protocol is discussed in Section 2.2, which provides essential information about the behaviour of network devices.

2.1 Network Architecture

The TCP/IP model is a collection of protocols that describe mechanisms used in network communication. Functions that the model provides are broken up into layers, where lower layers provide services to the layers above them [22]. This approach prevents protocols from implementing the same functionality repeatedly and introduces a hierarchy to the protocols.

The top layers handle data encoding, session management and application specific requirements. The lower layers handle routing and sending the data across the network. Before a packet traverses the network several steps are taken, in a process called encapsulation, that add necessary information for each of the layers to the data [32]. The process of encapsulation is described by Figure 2.1. The reverse process is called decapsulation and is performed when a packet is received. Layers and their functions in the TCP/IP stack are described as follows:

Application layer – This layer provides functionality for a specific application that is running on the computer. It does not define the application itself, but mechanisms that the application needs to function in a networking context. Essentially, this layer creates a bridge between the application and the network in the lower layers. Typical representatives of protocols at this layer are HTTP, POP3 and IMAP [32].

Transport layer – The transport layer includes multiple protocols, however, the majority of the traffic uses the TCP and UDP protocols. They provide error recovery or best effort delivery to the application layer respectively. The TCP protocol also provides congestion control by breaking up longer messages into shorter segments, so that the transmission speeds can be adjusted accordingly when the network is congested [22].

Network layer – This layer includes different protocols; the most commonly used protocols are IPv4 and IPv6, that are used for addressing the traffic by devices called routers. In order to deliver a packet to the correct recipient, it is necessary for each device to have a unique IP address. However, the address space of the IPv4 and IPv6 protocols is limited. Therefore, the IPv4 address space, which is still the most commonly used protocol on layer 3, is almost depleted and new IP addresses are assigned in a limited manner [36]. Since the problem of the address space depletion has been apparent years in advance, a mechanism to remap multiple IP addresses into one, called Network Address Translation (NAT), has been implemented. However, by implementing NAT the uniqueness of the IP address of a device has been lost, and multiple devices are using the same IPv4 address.

Data link layer – The data link layer describes protocols and hardware necessary for network communication. This layer introduces a unique address to networking interface cards (NICs) that is burned into the hardware. Addresses comprise of two parts - the first 24 bits is the manufacturer identifier, with the remaining 24 bits comprising the serial number of the NIC [32]. In theory each of the MAC addresses is unique, however, there exist duplicates due to the possibility to rewrite them in the operating system, or failures of manufacturers to assign unique MAC addresses.

Physical layer – The first layer is comprised of standards defining the transportation medium such as connectors, pins, light modulations and procedures on how to use the physical medium [32].

Based on the aforementioned paragraphs, it is possible to use the combination of the MAC addresses and the IP addresses as a unique identifier of a device on a network. Examples of these addresses can be seen in Table 2.1.

Address Name	Example Address
IPv4 address	192.168.1.1
IPv6 address	2001:0db8:85a3:0000:0000:8a2e:0370:7777
MAC address	00:0a:95:9d:68:aa

Table 2.1: Examples of addresses used in networking.

2.2 NetFlow

NetFlow is a Cisco proprietary protocol that was originally designed for network billing and accounting, however nowadays, is mostly used by network administrators to monitor network traffic. In recent years, this protocol has seen a rise in providing anomaly detection and investigative capabilities [38] that are often used by network administrators.

NetFlow can be utilized on all layers of the network and used for network data collection in various deployment scenarios. However, the best practice dictates that NetFlow should be enabled as close to the wide area network as possible. The implementation of the protocol relies on three different components [38, 4]:

- **NetFlow Collector** – Is a physical or virtual device that collects data from infrastructure devices. The Collector is also responsible for storing flow records.
- **NetFlow Exporter** – Is a physical or virtual device that is responsible for collection and pre-processing of NetFlow data.
- **NetFlow Analyzer** – Is a web-based or a desktop application used for analyzing NetFlow data and creating statistics of network traffic.

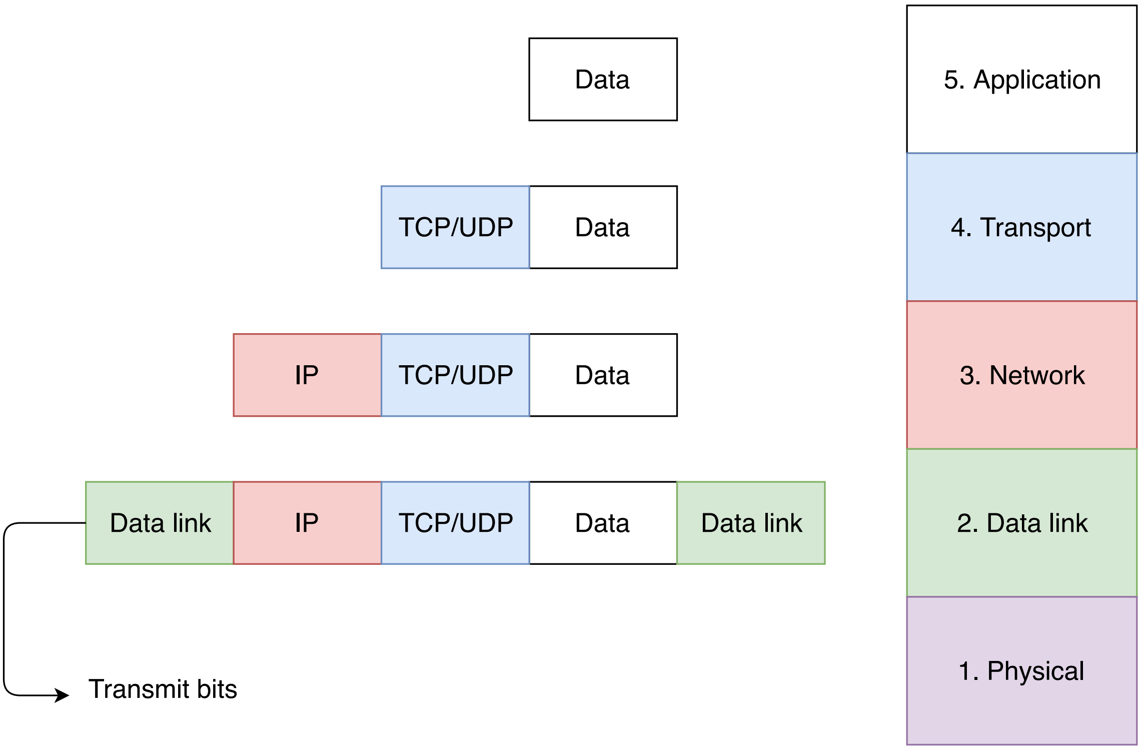


Figure 2.1: Steps taken in data encapsulation. Each new field represents additional headers that contain necessary information for each of the layers [32].

All of the packets flowing through a NetFlow enabled interface, also called an observation point, with the same identifiers are grouped together into a flow. "*A flow is a unidirectional series of packets between a given source and destination. Flows share the same source and destination IP addresses, source and destination ports, and IP protocol. This is often referred to as the five-tuple.*" [35] All of these flows are then aggregated to a database of NetFlow information that is also called the NetFlow cache. There are two types of cache:

- **Temporary cache** – Timers are used to keep track of the age of the data stored within the cache. Contents of the cache are sent to the Exporter after the timer expires. This type of cache is usually used to keep track of partial flows that can be used to detect threats on the network.
- **Permanent cache** – This type of cache is used to store flows permanently and is usually used for billing and accounting.

A set of observation points that can be aggregated into a single set of flow information is called an observation domain. A typical example of an observation domain is a set of interfaces on a router that are connected to the local area network (shortly LAN).

NetFlow has been implemented using the UDP protocol for efficiency, since a loss of packets does not have a large impact on the statistics provided by the protocol. However, this protocol is not a congestion-aware protocol, thus in a congestion-sensitive environment the Exporter and Collector have to use a dedicated link between each other. In the case that the link between the Exporter and Collector is not within one hop, it is necessary to ensure that the link is able to handle the maximum bursts of network traffic from the Exporter.

There have been several versions of the protocol since its initial release in 1996 and various versions are used on devices depending on the model and the version of the operating system. The most commonly used version of NetFlow is Version 9. This version is based on templates that provide a flexible design for future enhancements that are discussed later in this section. The header, which is of a constant size, is specified in Figure 2.2.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Version = 9               |          Count = 7            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           sysUpTime                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           UNIX Secs                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Source ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.2: NetFlow Version 9 packet header format [4].

- **Version** – NetFlow version and export format. Currently version 9 is used.
- **Count** – Number of flows exported within this packet.
- **sysUpTime** – Time since the device was booted in milliseconds.
- **UNIX Secs** – Time of the export packet leaving the Exporter in seconds since January 1st 1970.
- **Sequence Number** – A counter of Export Packets sent by the Exporter from an Observation Domain. This value is cumulative, and is used by the Collector to identify if any of the Export Packets have been missed or not [4].
- **Source ID** – A 32-bit identifier of the Exporter Observation Domain. NetFlow Collectors use the combination of the source IP address and the Source ID field to separate different export streams originating from the same Exporter [4].

One of the major advantages of NetFlow Version 9 compared to the previous generations, is the flexibility of the flow record format. In the past, any changes in data collection or field additions to the protocol required a new version to be released. With the addition of templates, new types of flow records can be defined without any changes to the structure of the export format. However, the most important template, FlowSet, from which the data is used in this thesis, is specified in Figure 2.3.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       FlowSet ID = 0          |      Length = 28 bytes        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Template ID 256         |       Field Count = 5         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     IP_SRC_ADDR = 8           |       Field Length = 4        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     IP_DST_ADDR = 12          |       Field Length = 4        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     IP_NEXT_HOP = 15          |       Field Length = 4        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       IN_PKTS = 2             |       Field Length = 4        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       IN_BYTES = 1            |       Field Length = 4        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2.3: NetFlow Version 9 template FlowSet format [4].

- **FlowSet ID** – FlowSet ID is a 16-bit unsigned integer used to distinguish different types of FlowSets. FlowSet IDs from 0 up to 255 are reserved for special FlowSets; IDs greater than 255 are Data FlowSets.
- **Length** – Total length of the data of the FlowSet.
- **Template ID** – Unique template ID for each new Template Record. The template ID is unique only to the observation domain.
- **Field Count** – Number of fields in a template.
- **IP_SRC_ADDR** – IPv4 source address of the flow.
- **Field length** – Length of the previous field.
- **IP_DST_ADDR** – IPv4 destination address of the flow.
- **IP_NEXT_HOP** – IPv4 next hop address of a flow.
- **IN_PKTS** – Number of packets associated with a specific flow.
- **IN_BYTES** – Number of bytes in a flow.

2.3 HTTP Fingerprinting

Devices often show a certain deviation from one to another even within the same model, which makes each of these devices unique. This unique set of attributes is called a device fingerprint. Fingerprinting is a common practice in network security, as it facilitates the collection of network device fingerprints.

Similarly to these devices, each web browser contains a certain degree of uniqueness and research has proven that it is possible to identify users with a degree of certainty [13]. The most commonly used method to re-identify users revisiting a website is through cookies. However, cookies pose a threat to privacy, as an attacker can abuse this information and authenticate as their victim. Nowadays, users are more aware of the threat cookies pose to privacy and users often delete or block the access of cookies. When a user blocks cookies and the server wants to set a cookie, the resulting error is detectable by the server and can be used as a parameter in the fingerprint.

However, users are very rarely aware of super-cookies [33] that are used by Internet service providers or other companies to track visited websites along with the number of their visits. These cookies can also access information contained within traditional tracking cookies, such as login information and cached images.

Among the most common headers used to identify the application type, operating system, language, software version and other information, is the *User Agent header*. This header contains information about the types of content that are understood by the client, which are described by MIME types. MIME types need to be set by the browser differently for each context. Data regarding the use of an Ad blocker or plugins to prevent tracking contained within the Do Not Track HTTP header can also be used to create a unique set of attributes to form a fingerprint. However, these attributes are not stable and change quite frequently, but algorithms such as the one in the paper from Eckersley [13] are able to track these changes.

Information about the browser can be collected not only from HTTP headers but also using AJAX. AJAX is based on JavaScript and runs within the browser. It is used to send information from the user to the server and back asynchronously. Information, such as installed fonts and screen resolution, can be obtained using this method.

Based on the information that was mentioned in the previous paragraphs, we can create a matrix where all of the aforementioned information is aggregated. Each row of this matrix corresponds to one flow of communication in one direction and in combination with the source, destination IP addresses and ports from flows that are described in Section 2.2. However sometimes, some of the information is not available and the data in these fields needs to be added, or the entry needs to be entirely removed from the data-set in order to produce accurate results. Techniques that are used for data preparation are described in Section 5.2.

In order for the HTTP protocol to work correctly, it is necessary to use the domain name service protocol as well. The main goal of domain names is to provide easy to remember names for the users to associate with different hosts, networks and administrative organizations. The domain name service (shortly DNS) is a service that provides translations from domain names to IP addresses [22]. Thus, if a user asks for the host IP address or email information of a server, an appropriate query is passed to the resolver with the requested domain name. Programs usually interact with the domain name space using a resolver, where a typical use case scenario is described by the Figure 2.4.

Attribute	Example Values
User-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.120 Safari/537.36
Cookies enabled	True/False
Do Not Track/Ad Blocker	True/False
Screen resolution	3840x2160@60Hz
Browser plugins	AdBlock Google Docs Offline WIS Floating Clock
MIME types	video/webm;codecs=h264,vp9,opus video/x-matroska;codecs=avc1 audio/webm audio/webm;codecs=opus application/pdf
Language	en-us,en;q=0.5
Fonts	Times New Roman
Timezone	CET(GMT +1)

Table 2.2: Fingerprint examples.

```
             Local Host                        |  Foreign
                                               |
+---------+               +----------+         |  +--------+
|         | user queries  |          |queries  |  |        |
|  User   |-------------->|          |---------|->|Foreign |
| Program |               | Resolver |         |  |  Name  |
|         |<--------------|          |<--------|--| Server |
|         | user responses|          |responses|  |        |
+---------+               +----------+         |  +--------+
                            |     a~           |
            cache additions |     | references |
                            V     |            |
                          +----------+         |
                          |  cache   |         |
                          +----------+         |
```

Figure 2.4: An example of a DNS resolver scenario [30]. The user program requests an IP address using a domain name, and the resolver either sends the query to another DNS server or sends a response from its cache.

The behavior described in the Figure 2.4 enables network administrators to collect data and create user profiles that become more accurate over time. The user re-identification of Kumpost [21] uses HTTP(S) and SSH connections in combination with DNS queries to re-identify users. Kumpost's user profiles are based on sparse access frequency vectors combined with the number of connections to each destination IP address. The proposed method has been tested on aggregated monthly NetFlow logs and achieves an accuracy of 78.3% with SSH traffic.

2.4 TLS Fingerprinting

Transport Layer Security protocol (shortly TLS) provides a secure channel between two communicating applications. The protocol itself provides authentication, confidentiality and integrity [37]. Authentication is always guaranteed from the server side and is optional from the side of the client application. Authentication is provided using asymmetric cryptographic algorithms, such as RSA, ECDSA and PSK. Confidentiality is guaranteed by using the channel that is established before data is sent and is only visible to the endpoints. Lastly, integrity is guaranteed by the use of an encrypted channel, where any changes by the attacker can be detected by the server or the client, depending on the direction of the traffic.

TLS is composed of the handshake protocol and the record protocol. The handshake protocol handles authentication, cryptographic mode negotiation and establishes keys. The TLS handshake consists of the three steps described in Figure 2.5. The record protocol provides security by splitting the traffic into a series of records that are protected using individual keys [37].

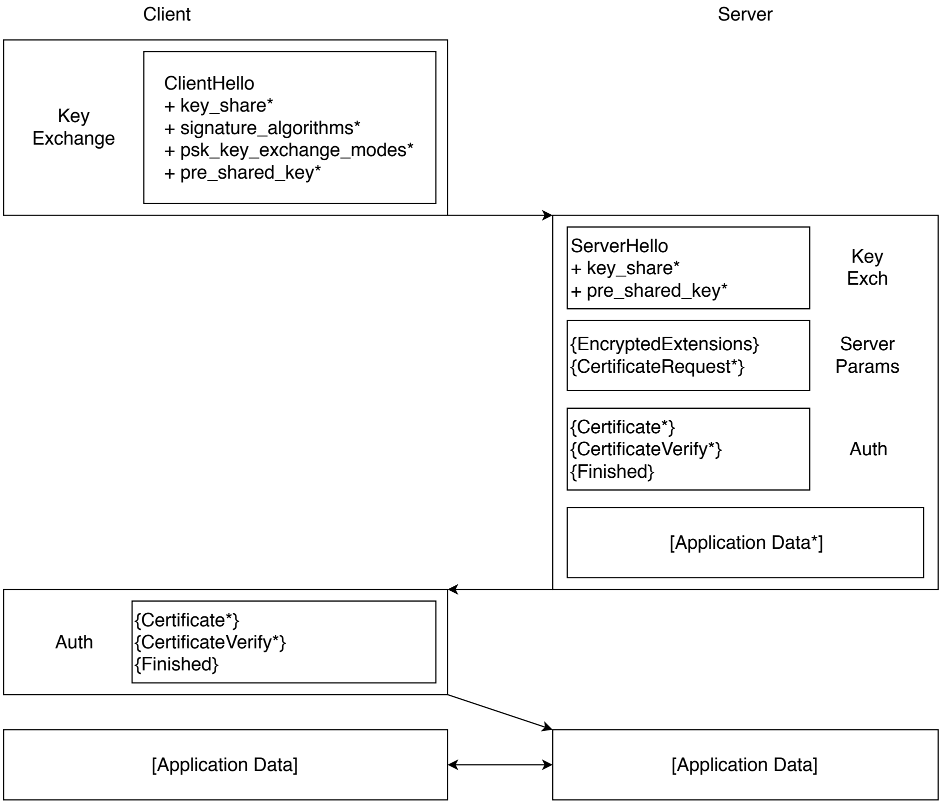


Figure 2.5: Scheme of the TLS handshake taken from [37]. + denotes extensions to the messages. * denotes optional or situation dependent fields. *{}* and *[]* denote messages protected by keys derived from the *handshake_traffic_secret* and *handshake_traffic_secret_N* respectively.

Handshakes are resistant to tampering, however, fingerprints can be created by inspecting the *ClientHello* messages. By aggregating this information to a database it is possible to collect fingerprints of malicious software and vulnerable applications [2]. This information can be used in combination with the information collected by the NetFlow or IPFIX protocols, described in Section 2.2, to identify network devices. This approach has been proven to be very effective by Anderson et al. by achieving an accuracy of over 90%, depending on the technique used [3]. The information collected from the TLS handshake is stored in the JSON format and the meaning of the fields is explained in Table 2.3.

Field	**Description**
id	ID of TLS fingerprint.
desc	Description of the TLS fingerprint. Usually in the form of an application name or URL.
record_tls_version	Version of the TLS record protocol.
tls_version	Version of the TLS protocol ranging from the now deprecated SSLv2 to the newest TLSv1.3
session_id	ID of TLS session
ciphersuite_length	Length of key used in the cryptographic algorithm.
ciphersuite	Type of cryptographic algorithm used.
compression_length	Size of compressed data in bytes.
compression	Type of compression algorithm used. Since TLS version 1.3 no compression is used, however, older versions of the protocol are still widely used.
extensions	Extensions used by the TLS protocol. For further information see Figure 2.5.
e_curves	Ephemeral Elliptic Curve Diffie-Hellman key agreement.
sig_alg	Type of signature algorithm used.
ec_point_fmt	Ephemeral Elliptic Curve Diffie-Hellman key format.

Table 2.3: Table describing the fields that are collected in the process of TLS fingerprinting [6].

2.5 Clock-skew

Computers measure time using crystal oscillators that use mechanical resonance of a vibrating crystal in a piezoelectric material to create a signal with a precise frequency. Frequencies of these oscillators depend on the cut angle and type of the crystal. Manufacturing of the crystals has some tolerances that introduce differences to the produced crystals resulting in slightly different frequencies. Differences in frequencies create a phenomenon called clock-skew that provides a method to reliably identify devices [23]. Clock-skew is commonly given as a measure in parts per million (shortly ppm), which means that per every million time units the error is *n-time units*. Clock-skew is unique and reported to be reliable for identifying users in TCP sessions according to Kohno et al [19].

Each packet that is sent over the network is delayed by n *time units* from the time the packet was sent and consequently received by another device. These delays $\varepsilon(t)$ are caused by the time needed for processing the packet by the sender and observer. Delays, however, are not constant and are dependent on many factors. The following Figure 2.6 depicts the delay between the hosts.

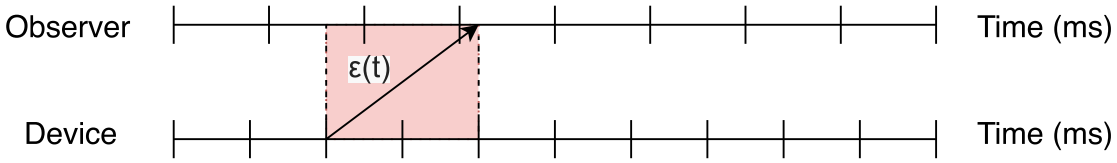


Figure 2.6: Clock-skew caused by the delays during processing and sending the data.

Kohno et al. proposed to estimate clock-skew by using the slope of the offset points, which is the difference from the beginning of the measurement (e.g., packet transmission) until the observer receives the packet. They have shown that the slope of the upper bound is similar to the slopes of the offsets.

Most of the timestamps used in the aforementioned research are from TCP and ICMP traffic. However, TCP traffic is not the only possible source of timestamps. HTTP servers can also generate timestamps, which show when the web page has been generated. The resolution of this timestamp is lower - only 1 second [34]. In order to compensate for this error, it is necessary to perform synchronized sampling [31].

Synchronized sampling is a process where the observer attempts to synchronize its clock to the clock of the HTTP server by sending HTTP requests in sync to the server. Some other application layer protocols can also send timestamps, however, their resolution is low as well and needs to be compensated for using synchronized sampling.

Chapter 3

Classification Methods and Model Validation

Classification is a process where each object is assigned to a class based on its attributes, also called features. Before any object is assigned to a class, a model needs to be created. These models describing the data are called classifiers. Prior to a classifier being able to make predictions, it needs to be trained. Training is performed on input data that contains labels of which class a feature vector belongs to. Afterwards, classifiers are able to make predictions on previously unseen unlabeled data. This approach provides more flexibility in the creation and evaluation models, which proves to be more robust compared to traditional tracking algorithms. Robustness provided by the models is a significant plus, since the datasets can vary in quality and availability of features. The first part of this chapter is dedicated to machine learning algorithms that are going to be used in the process of identifying network devices based on their behaviour. The second part of this chapter (Section 3.6) describes methods that verify the accuracy of created models by training models on a subset of data and verifying the results on another disjoint set of data.

3.1 Decision Tree Classifier

Decision trees are simple to understand models that used for partitioning the input space into regions, where a different model can be applied to each of these regions [5]. The division of regions can be interpreted as a sequence of binary decisions that are applied to the input features. Each leaf subsequently corresponds to a class. An example of a decision tree which decides whether to go on a hike or not can be seen in Figure 3.1.

During the training phase, it is essential to determine the features, and the threshold values of these features, that will be used to create the structure of the tree. Training starts with choosing the feature which is used to split the data into regions with the lowest cost to accuracy. Afterwards, the algorithm continues in the same manner for the newly created leaf nodes recursively until the process of building the decision tree is finished.

One of the major advantages of this model compared to others, is the simplicity to visualize and understand the model. Decision trees can handle numerical and categorical data as well. However, one of the disadvantages is the susceptibility to over-fitting. Overfitting manifests itself by creating nodes that lower the accuracy of the classifier. In order to prevent over-fitting, methods such as *prepruning* and *postpruning* are introduced [46, 16]:

- **Prepruning** – Nodes are not created during the process of building the tree. In case of the branch not reducing the error of predictions significantly, that branch is removed and replaced by a leaf node corresponding to the largest number of samples in that specific branch of the tree. The advantage compared to other methods is the low computational complexity.
- **Postpruning** – “Unnecessary” nodes are removed after the creation of the tree. This method makes the model more accurate compared to *prepruning*, however, the computational cost is higher.

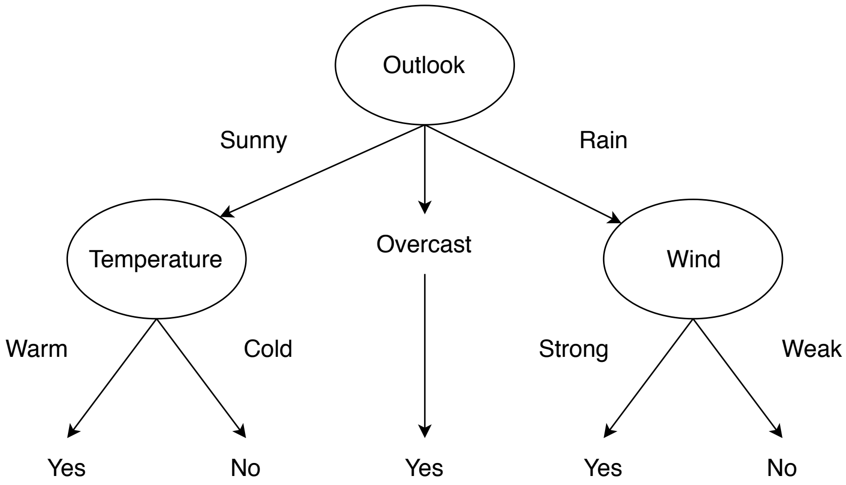


Figure 3.1: Example of a decision tree deciding whether to go on a hike or not.

3.2 k-Nearest Neighbors

The k-Nearest Neighbor (k-NN) is a supervised machine learning algorithm that is simple to understand and can be used for either classification or regression. This algorithm belongs into the category of lazy learning algorithms, or local learners, which work under the assumption that similar classes of data lie in a close proximity to each other [40]. This means that the algorithm does not need to create a model to predict data, meaning there is no training phase. The algorithm calculates distances based on different metrics depending on the use case.

The most commonly used distance metric is Euclidean distance [12], defined by Equation 3.1. However, Euclidean distance is not an appropriate distance metric in a high dimensional space, since it provides a poor contrast between the furthest and nearest neighbor [1]. Among other examples of distance metrics used in this thesis are the Minkowski distance [14] defined by Equation 3.2, Dice similarity coefficient [48] defined by Equation 3.3, and cosine similarity [44] defined by Equation 3.4. Cosine similarity is a metric that is most often used in text similarity and plagiarism detection, where it performs well in a high dimensional space. The process of classification using the k-NN algorithm can be summarized into three steps [40, 15]:

1. Given a query point x^*, find k nearest neighbors from the data-set $D = \{(x_i, y_i\}$, where x_i is the input data and y_i is the label

2. Calculate the distances between x^* and x_i
3. Return the mode of the k labels chosen based on the closest distance between x^* and x_i

$$d_n(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.1}$$

$$d(i, j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + ... + |x_{ip} - x_{jp}|^h} \tag{3.2}$$

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{3.3}$$

$$similarity = cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.4}$$

The choice of the correct value for k is data dependent. It is recommended to test various values to find the suitable k for the given data-set. In cases where the value of k is too small, the predictions are susceptible to noise. On the other hand, when k is too large, the boundaries between classes become less distinct [15].

The major advantage of this algorithm is its simplicity and ease of implementation. However, the memory requirements are large since the entire data-set needs to be in the memory in order to classify, which makes it complicated to work on large data-sets.

3.3 Naive Bayes Classifier

The Naive Bayes classifier belongs to the family of probabilistic classifiers. This algorithm uses the counts of individual features and their combinations from which it subsequently calculates probabilities of occurrences. However, this algorithm also assumes that each and every one of the features is independent, which is rare in a real-world setting. Even though the assumptions for this method are not always true, the Naive Bayes classifier performs well in various supervised classification settings. The basis of the classifier is the Bayes theorem described by Equation 3.5, where A, B are events, *P(A/B)* is the probability of A given B is true, *P(B/A)* is the probability of B given A is true and *P(A), P(B)* are independent probabilities of A and B [40].

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.5}$$

Depending on the application there are three different variants of the algorithm available:

- **Binomial Naive Bayes Classifier** – This variant of the classifier assumes a binomial distribution over the data-set. It is mostly used in a context where it is necessary to classify between two classes, which is most often used for spam detection.
- **Multinomial Naive Bayes Classifier** – This variant of the classifier assumes a multinomial distribution over the data-set. It is used when there are multiple classes for classification. The inputs are feature vectors, where the probabilities for each of the classes are increased with each observation.

- **Gaussian Naive Bayes Classifier** – The Gaussian Naive Bayes classifier assumes a Gaussian distribution over the data-set. This version is used mostly when continuous values are distributed according to the normal distribution.

The training phase consists of deriving the conditional probabilities from the data-set. This proves to be a big advantage compared to other methods, since it is not necessary to add any other parameters during the training phase. However, this type of classifier is sensitive to imbalanced data-sets, so it is necessary to provide an appropriate data-set to achieve best possible results [25].

3.4 Term Frequency and Inverse Document Frequency

Term frequency and inverse document frequency (shortly tf-idf) is a statistical method designed to evaluate the relevance of a term in a document, or multiple documents. This method is most commonly used in text mining, text similarity and clustering.

The weights of terms for a document are determined by the number of occurrences of the given term in a document, and is considered to be a quantitative representation of a document [26]. From this point of view, a document is called a *bag of words* model, since the order of the terms does not play a role in the model. Therefore, the semantic difference between the sentences *The fox has eaten the rabbit* and *The rabbit has eaten the fox* are lost and the model for these two sentences is identical. However, this shortcoming of the method plays a role in this thesis, as the order of the features influences their semantic meaning.

There is a low chance that the source and destination IP address might be the same when aggregating flows from a certain time window, however, their semantic meaning is different. This phenomenon might be caused by a simultaneous DHCP lease termination by two network devices, and afterwards, their leased IP addresses are exchanged. If these two devices communicate within the same time window as their IP addresses change, these values would appear to have a similar weight, however, their semantic meaning is different. Therefore, these two features are analyzed separately and appended at the end of the analysis phase to the rest of the term frequency vector.

A major downfall of term frequency is that all of the terms are considered to be equally important across all of the documents. Therefore, document frequency is introduced in order to be able to discriminate between the relevance of two documents to a query. The document frequency expresses the number of documents that contain a given term from the entire set.

Lastly, the inverse document frequency (shortly idf) is introduced. This value expresses how common or rare a term is in the document. The *idf* is a logarithmically scaled function defined by the following Equation 3.6, where *df* represents the document frequency.

$$idf = \log \frac{N}{df} \tag{3.6}$$

Lastly, the value tf-idf is calculated using the following Equation 3.7:

$$tf\_idf = tf * idf \tag{3.7}$$

The values that are assigned to each of the terms can be summarized into the following points [26]:

1. Highest values of *tf_idf* is assigned to the terms, when the given term has many appearances in a short document

2. Lower values of *tf_idf* are assigned to terms that are contained in many documents or are rarer within one document

3. The lowest values are assigned to terms that appear often in a large majority of documents, therefore, do not have a high informational value

3.5 Outlier Detection

This section describes methods used to detect outliers. Outliers are data points that are distant from the overall pattern of observations of one class. These data points are under normal circumstances considered as values that negatively impact the accuracy of the created model and skew the results towards incorrect predictions [42]. Therefore, in statistics these data points are often removed from the data-set to increase the accuracy of the model. However, in the case of this thesis, outlier detection can be used to detect previously unseen devices, since the calculated similarities, or distances, described in Section 3.2 will be outside of the pattern of the rest of the data points. This section describes two methods used for outlier detection used in this work. The first one is outlier detection using z-score values described in Section 3.5.1 and the second is new device detection using the k-Nearest Neighbors algorithm described in Section 3.5.2.

3.5.1 z-score Outlier Detection

The normal distribution or the Gaussian distribution, also known as the bell curve, is one of the most important distributions that copies many natural phenomena. A graphical representation of the normal curve can be seen in Figure 3.2. Parameters that are commonly used to describe the normal distribution are the mean μ and standard deviation σ [42]. These values represent the center of the distribution and how much the data varies (eg., the spread of the distribution).

A common practice when comparing multiple different distributions is through standardization. A standardized variable has a mean $\mu = 0$ and a standard deviation $\sigma = 1$. To calculate the standardized normally distributed variable (e.g., the z-score), the following Equation 3.8 is used. The z-score essentially represents how many standard deviations the given point is from the mean μ.

$$z = \frac{x - \mu}{\sigma} \tag{3.8}$$

From empirical observations when a distribution is roughly under the normal curve, it is true that [42] (graphical representation can be seen in Figure 3.2):

- Approximately 68% of all observations of a variable lie within one standard deviation ($z = \pm 1$) from the mean μ

- Approximately 95% of all observations of a variable lie within two standard deviations ($z = \pm 2$) from the mean μ

- Approximately 99.7% of all observations of a variable lie within three standard deviations ($z = \pm 3$) from the mean μ

However, this empirical observation can also be used as a method to detect outliers. Based on the aforementioned empirical rule, all of the data belonging to this distribution should be located within three standard deviations from the mean. The data points that are further than 3 standard deviations from the mean μ (e.g., $z = \pm 3$) are considered to be outliers [39]. These data points, in the case of this thesis, do not skew the data and introduce interference, however, they are considered to be a previously unseen device. The details in how this fact is used are further discussed in Section 4.2.

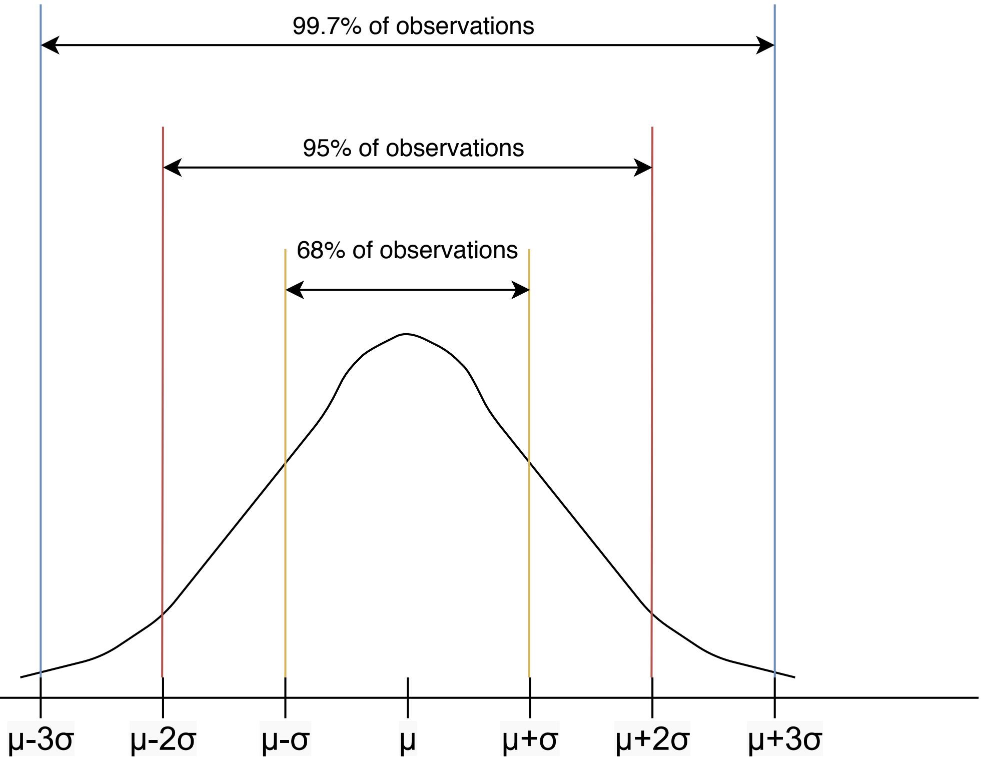


Figure 3.2: Graphical representation of the empirical 68-95-99.7% rule. The depiction of the empirical rule is inspired by the book [42].

3.5.2 Outlier Detection Using k-NN

The algorithm described in Section 3.2 can also be used for outlier detection, among other use cases. This approach belongs into the family of supervised methods, which are good at detecting data points that are located in regions with low density of data [47]. Figure 3.3 illustrates the case where data is clustered into one area and the rest of the data points outside of this region are considered to be outliers.

The algorithm uses the Minkowski distance defined by Equation 3.2 by default, however, the distance metric can be changed. After the distances are calculated, the values are remapped to the range [0, 1]. Lastly, the algorithm introduces a constant called `contamination`. This is the value which represents what proportion of data are outliers.

3.6 Model Validation

Over-fitting has many faces and is difficult to detect by simply evaluating the model's results. Therefore, there is a need for mechanisms that prevent over-fitting and create

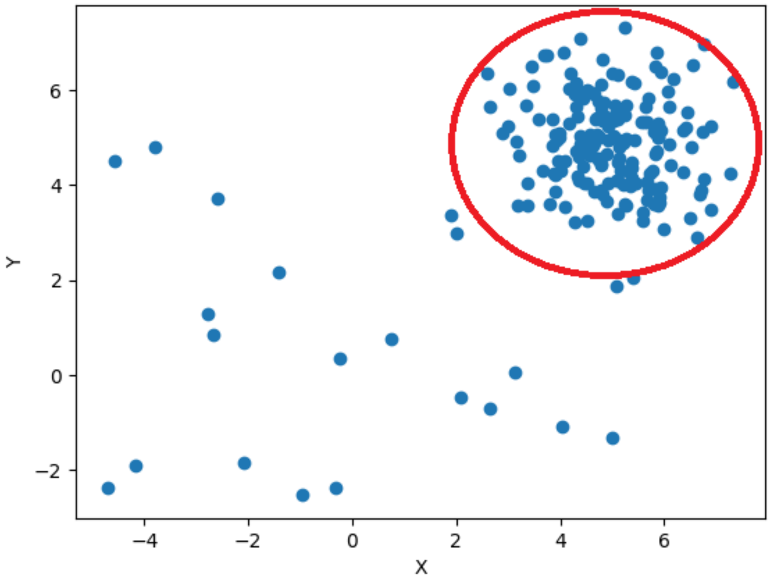


Figure 3.3: Illustration of data points of a phenomenon clustered in the circle in red. All the rest of the data points outside the marked area are considered to be outliers.

a better basis for the model to perform well on data that is different from the training data-set. This section introduces techniques used for model evaluation. The first part describes re-sampling algorithms that are used in this thesis to break up and evaluate the accuracy of the model on the data-set. Furthermore, this section describes methods of training to prevent over-fitting or under-fitting the model to the training data [11, 7, 40].

Over-fitting refers to either capturing the noise of the data-set, or fitting the model extremely well to the training data, and performing poorly on unseen data. Under-fitting refers to the case when the model does not recognize patterns of the training on unseen data and yields poor results.

k-Fold Cross-validation

Cross-validation is a procedure used to re-sample a data-set to yield more accurate results on unseen data on a limited data sample. This approach in return reduces the impact of over-fitting and under-fitting. k-Fold cross-validation is mostly used in a context where a model predicts classes into which the data belongs.

The procedure starts with randomly shuffling the data. Afterwards, the data is separated into k equally sized folds. Subsequently, the model is trained on *k-1* folds of data. The last fold is then used for evaluation of the accuracy of the used model. This process is repeated until each of the folds has been used as a validation and training fold [40]. A graphical representation of this algorithm with $k = 5$ folds can be found in Figure 3.4.

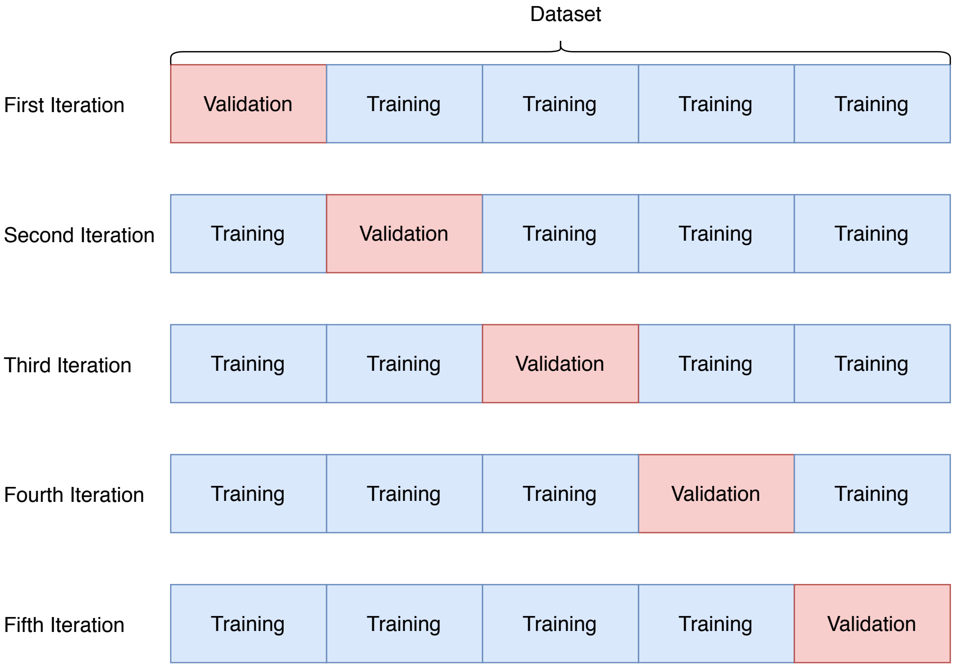


Figure 3.4: Graphical representation of the k-Fold cross-validation algorithm with $k = 5$. In each of the iterations a part of the data-set is used for training, and a part is used for validation to find the best model representing the data-set.

This method of verifying the model accuracy is used due to its simplicity, and results in a less optimistic and/or less biased estimate of the model. Empirically it has been shown that $k = 5$ or $k = 10$ yield the lowest test error estimates, suffering from neither excessively high bias nor high variance [11].

Bootstrapping and Bagging

Bootstrapping is a method commonly used in statistics for re-sampling a data-set. This technique samples the data with replacement of the data points. It is also commonly used in machine learning for evaluating the accuracy of a model on data that was not included in the data set [8, 40].

Samples of the data-set are chosen one at a time and returned back to the overall set to be chosen again. This allows a specific data point to appear multiple times within one data-set. These training sets are also called bootstrap samples. Afterwards, models $f_m(x)$ using methods mentioned in Section 3.3 are trained on the sampled data-set x and the process is repeated M *times*. Lastly, after all the models of the same type have been created, the average of these models is taken using the Equation 3.9. The averaging process is also called *bagging*.

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x) \tag{3.9}$$

Chapter 4

Proposed Algorithmic Solutions

This chapter describes two proposed approaches to network device identification that use a combination of machine learning algorithms and concepts used in networking. The algorithms proposed in this thesis are designed to overcome the limitations of fingerprinting methods, and provide a robust model for each of the devices using statistical methods and machine learning algorithms. The usage of these methods is essential since the data-sets do not always contain all of the features, therefore, the methods need to be able to overcome the lack of their availability. The initial Section 4.1 contains the the first algorithm that is inspired by text similarity methods and plagiarism detection. However, this algorithm has multiple downfalls that are described in the section. The following Section 4.2 contains a modified version of the aforementioned algorithm that overcomes the predecessor's limitations. The algorithm uses a sliding window to update the device profiles and is capable of detecting previously unseen devices appearing for the first time after the training phase. This method is the final and best performing algorithm to identify and track network devices based on their behaviour, and does not require to be retrained after a device is added to the network. It allows network administrators to reliably track the devices without the necessity of knowing any of the unique identifiers, such as the MAC address.

4.1 Nearest Neighbor Combined Model

Since the data-set contains tens of millions of flows, it becomes computationally difficult to classify each flow by itself. Therefore, it is necessary to aggregate the flows into a profile that will be stored and later compared to the current device behavior. As it is necessary to aggregate multiple flows into one profile, the nature of the data-set (and aggregated profiles) resembles written text.

One of the most common methods used to analyze text and compare similarity of texts is using frequency analysis in combination with a similarity measure, such as Jaccart or cosine similarity. Due to the resemblance of the aggregated flows to text, it is possible to create profiles using frequency analysis for these devices. Therefore, the exported data in the initial learning phase (the first hour) is aggregated into one large profile based on their MAC addresses and usernames. Afterwards, the profiles are aggregated in 5-minute time-windows based on the combination of the source IP address and username, since the goal of this thesis is to design an algorithm capable of identifying devices based on their behaviour, and MAC addresses are rarely available in a production environment. They are, however, used as labels for the purposes of this thesis. This combination of features (source IP address and username) to create device profiles has been chosen, since it provides a very simple and effective way to distinguish devices in a given time-window. However, this combination of features is not sufficient to identify a specific device, since a user can own multiple devices and the source IP address of a given device may change over time. An illustration of flow aggregation and frequency analysis on the data can be seen in Tables 4.1 and 4.2. The shorter time-window during the classification phase has been chosen due to the fact that networks that contain a large array of devices with a lot of movement of users, such as school networks, have a shorter DHCP lease, usually 5 minutes [18]. Therefore, the shorter time-window lowers the probability that the source IP address is shared with a different device, which could happen in a longer time-window and aggregate multiple devices into one profile for classification, which would result in an incorrect classification. An illustration of the method of how profiles are segmented can be seen in Figure 4.1.

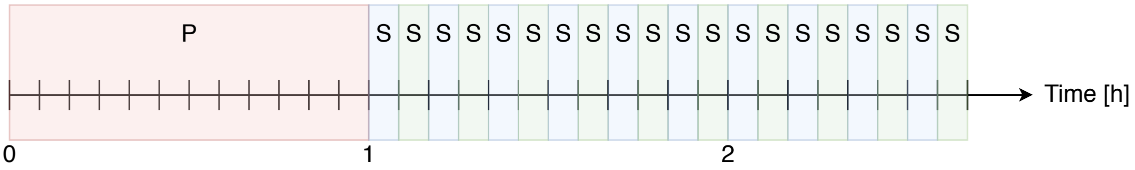


Figure 4.1: Illustration of timeline segmentation. *P* represents the initial profile that is created during the training phase. *S* represents profiles that are scored during the classification phase.

Flow 1	Flow 2	Flow 3	Flow 4
User 1	User 1	User 1	User 1
src IP A	src IP A	src IP A	src IP A
dst IP B	dst IP B	dst IP C	dst IP D
port 1	port 1	port 1	port 2
google.com	google.com	-	facebook.com
TLS 1	TLS 1	TLS 2	-
-	HTTP f 1	-	-

Table 4.1: Example of a 5-minute aggregated profile for a device based on its source IP address and username.

After the initial training phase is completed, the classification phase follows. Classification is performed by finding the closest neighbor to the previously created large device profile using cosine similarity. When a device is classified as one of the devices, the data from the aggregated 5 minute time window is added to the large profile to extend the features and increase the prediction accuracy in the future.

However, the approach described in the previous paragraphs does not take into the account new, previously unseen devices. Therefore, a method such as z-score outlier detection is necessary to detect a newly connected device, however, it is not explored due to the algorithms deficiencies described in Section 6.3. Another disadvantage of this approach is the size of the profiles. With every classification the size of the large profiles increases and over time the device profiles will not be able to be stored in the ram, mainly in large

Term	Frequency
src IP A	4
dst IP B	2
dst IP C	1
dst IP D	1
port 1	3
port 2	1
google.com	2
facebook.com	1
TLS 1	2
TLS 2	1
HTTP f 1	1
-	5

Table 4.2: Example of frequency analysis including only the features used in classification. The column frequency represents the vector used in similarity comparison.

corporate networks, which might contain over 10,000 devices. Therefore, the following Section 4.2 contains a description that is designed to overcome these challenges.

4.2 k-NN with Segmented Profiles Model

Similarly to the proposed algorithm in the previous section, the method described in the following paragraphs uses flow aggregation to create device profiles. However, this approach unifies the length of the time-windows and improves memory efficiency along with accuracy over a longer time period.

The algorithm proposed in this section segments the timeline into n-minute time windows called device profiles. Since these time-windows are shorter (less than 10 minutes), it enables the algorithm to aggregate flows in multiple approaches, based on the combination of the source IP address and username, or MAC address and username, even during the training phase, due to the fact that the probability of aggregating multiple devices into one profile due to the DHCP lease termination is lowered significantly. An illustration of the timeline segmentation can be seen in the Figure 4.2. The length of the n-minute time-window is evaluated in the Section 6.4 along with experiments with the rest of the parameters introduced in the algorithm described in this section.

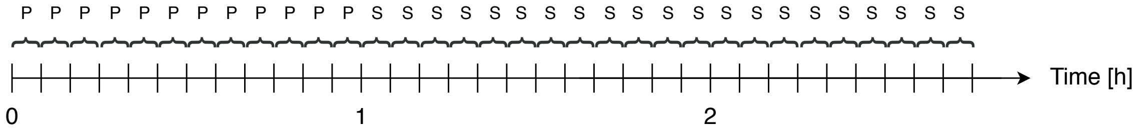

P P P P P P P P P P P P S S S S S S S S S S S S S S S S S S S S S
Time [h]
0
1
2

Figure 4.2: Illustration of how the timeline is segmented. P represents the initial profiles that are stored from the training phase. S represents profiles that are scored during the classification phase.

After the flow aggregation in a given n-minute time-window is completed, frequency analysis is performed using the same methodology as described in the previous Section 4.1, with an example in Tables 4.1 and 4.2.

Since it is necessary to be able to efficiently search for the usernames and their corresponding devices and their profiles, cascading hash tables have been chosen to improve searching efficiency. The following Figure 4.3 describes the system how device profiles are stored.

```
│
└── User 1
│   └── Device 1
│   │   └── [Profile 1, Profile 2, ...]
│   └── Device 2
│       └── [Profile 1, Profile 2]
└── User 2
│   └── Device 1
│       └── [Profile 1, ...]
└── ...
```

Figure 4.3: Hierarchy describing the structure of profiles stored in memory.

Some of the features in specific cases might contain duplicates of the same value but in a different context (such as source and destination IP addresses). This unlikely event in the case of a short time-window might occur when two devices that communicate among each other during the training phase terminate the DHCP lease in the same time window. Afterwards, they might be reassigned their IP addresses in reverse, which would cause the loss in directionality of the communication, which would not be detected by a traditional frequency analysis method. Therefore, the features source IP address and destination IP address are analyzed separately, and then appended to the vector containing frequencies of all of the features (an example of flow aggregation based on source IP address of a device can be seen Table 4.1 and Table 4.2). This approach enables the creation of user profiles that are memory efficient and accurate for the given n-minute window. Afterwards, it is possible to compare the similarity of these vectors in any of the following time-frames.

However, the behaviour of users on their devices might change over time in the cases such as administrative jobs with a wide array of responsibilities. Therefore, the proposed algorithm needs to take into the account these changes over time. A solution to this problem is a method called sliding window, where m device profiles are stored in the memory and the oldest profile is replaced by a newer one obtained when a device is classified as said device. In the case that there are fewer than m profiles stored in the memory the newest device profile is added to the array.

Classification and device tracking is performed using the k-NN algorithm with cosine similarity as the distance metric, which proved to be an efficient method of tracking devices for the algorithm described in the previous Section 4.1. Currently classified user profiles are evaluated only with the ones that correspond to the given username based on which the profile is aggregated to reduce the time of classification.

Lastly, the proposed method needs to be capable of identifying new, previously unseen devices in the data-set, create profiles for them and introduce the device profiles into a state where they could be identified by the k-NN algorithm. Therefore, the z-score and k-NN outlier detection algorithms are introduced. These algorithms enable the proposed method to identify previously unseen devices, however, this approach creates only one device profile for the new device which is not sufficient for the minimal $k = 3$ in k-NN to identify a device as said device. To overcome this limitation, similarity values have been analyzed and a threshold has been determined, that is considered to be a high enough similarity to classify the current profile as the device with the highest similarity. Therefore, devices with exactly one similarity obtained from the k-NN algorithm above said threshold are classified as said device. In cases where all of the similarity values are below the threshold, the outlier detection algorithms are applied. In all other cases the traditional k-NN algorithm is used for the classification. After the introduction of this last improvement, even devices with only one profile created for them can be re-identified and tracked using the method described in this section.

Experimental evaluation with differing values of parameters of the algorithm are discussed in the Section 6.4. The values of the parameters with the best results are summarized in the Appendix B.

Chapter 5

Implementation and Design of the Tool

This chapter contains a detailed description of the tool containing two existing machine learning algorithms (descriptions in Sections 3.1 and 3.3) and two newly proposed network device identification methods described in Sections 4.1 and 4.2. This tool was created to encapsulate all of the tested methods throughout this thesis and provide an easy method to create experiments. The knowledge gained from the experiments using this tool can be transferred into a real life application, such as network device tracking at a university or a company. Each of the following sections describes a part of the design and explains its functionality.

Since the data-sets contain a vast amount of information, it is necessary to gain insight into the data. Therefore, the first part of this tool contains a statistical module that analyzes the behaviour of the devices and the variability of the fields. A detailed description of this module can be found in Section 5.1. Afterwards, it is necessary to remove any outliers - such as devices that change their source IP address over 490,000 times over an eight hour period. Methods used for filtering the data are explained in Section 5.2.

This thesis introduces multiple types of network device identification methods. They can be separated into two categories. The first one are methods that are trained beforehand and are used to identify devices that have already appeared in the training data-set. However in return, these methods classify new devices as one of the devices that has already appeared in the data-set beforehand. Therefore, this thesis proposes multiple methods that overcome this problem and do not need to be retrained when a new device is added to the network. However, these methods compare the similarity of behaviour of the currently classified device to previously created user profiles. Both of these types of methods are marked into which category they belong in Section 5.3. Their technical details are described in Chapter 4, and Chapter 6 contains their performance and tested settings.

Users have their own habits of visiting certain websites or using specific services during the day. Since these habits have a tendency to be repetitive in an office environment, it is possible to notice patterns in user behaviour as can be seen in Figure 5.1 [20]. Furthermore, these patterns can be tracked for each device using identifiers introduced in Section 2.1.

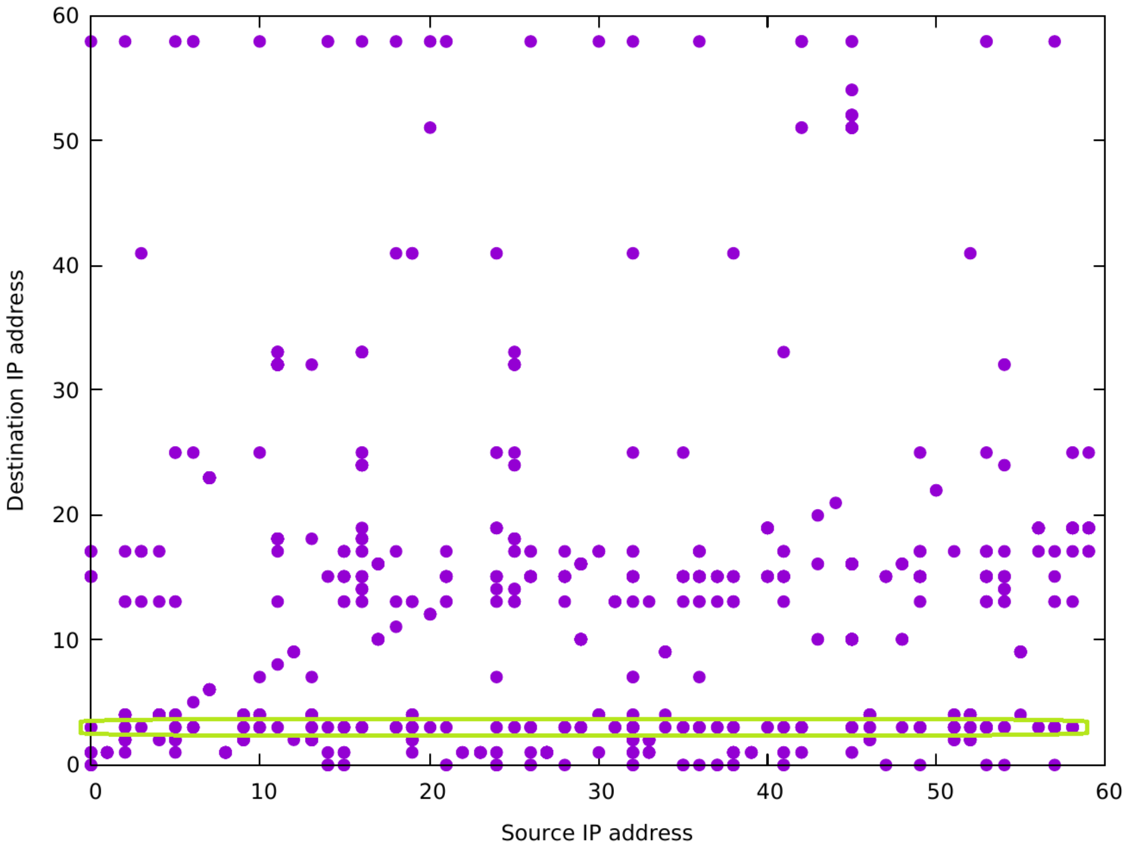


Figure 5.1: Scatter plot representing the source IP address on the X-axis and destination IP address on the Y-axis. The points show whether communication between two given IP addresses has taken place. Multiple dots located in one row indicate that there is a service running on the given destination IP address which can be seen in the case of destination IP address 4 (marked green).

5.1 Statistical Module

Statistics[1] is a very powerful tool in gaining knowledge about large data-sets, for example, the frequencies of changes, but also for detecting data points that are outliers and only introduce noise into the data-set. Since the data-set provided by Cisco Systems contains a large array of data it is necessary to analyze the provided data, separate the data into data-sets of manageable size, and remove any outliers within the data-set based on the statistical knowledge provided by this module.

Some of the attributes within the data-set can have a unique value in each instance, however, others should have a low variability or remain constant. Therefore, multiple statistical indicators are printed during the analytical phase such as the count, average, median, standard deviation, and the five-number summary [42]. The Table 5.1 summarizes the statistical data provided by the tool. Lastly, this tool creates a histogram of ports used by the devices within the data-set.

Column Name	**Statistical Information [Units]**
Source IP address	Average changes per device [Count] Maximum changes of a device [Count] Minimum changes of a device [Count] Median changes per device [Count] 25^{th} percentile changes [Count] 75^{th} percentile changes [Count] Average time between changes of a device [s] Median time between changes of a device [s] Maximum time between changes of a device [s]
Destination IP address	Unique destination visited by devices [Count]
MAC address	Unique devices [Count]
URL	Availability [%]
Port	Histogram [Count]
TLS Fingerprint	Availability [%]
HTTP Fingerprint	Availability [%]

Table 5.1: Statistical information given by the proposed tool about the columns in their respective units.

Based on the aforementioned information, it is possible to infer what values do the data points acquire and create a picture about the behaviour of devices within the data-set. Afterwards, it is possible to identify what devices in the data-sets are outliers from the expected behaviour and filter them out to reduce the noise contained within the data. Techniques that are used in this thesis for filtering the data and further transformations are introduced in Section 5.2. The data-set descriptions that are used in this thesis to evaluate the proposed methods are available in Appendix A.

5.2 Input Pre-processing

Large data-sets that have different origins have a higher risk of containing missing fields, inconsistencies or interference. This type of data, however, leads to a lower quality model

[1] https://www.britannica.com/science/statistics

and lower quality predictions. Therefore, it is important to ensure the data-set is of a high quality before it is used for training, and later, evaluation of new data points.

Before any filtering takes place, it is necessary to gain insight into the data. Techniques that are used to acquire the knowledge to prepare the data for the training phase are described in Section 5.1. From this statistical knowledge it is possible to identify outliers. Devices that change their source IP address more than a couple of times a second are considered outliers, since the expected time to change the source IP address should be in the magnitude of minutes or hours. Fortunately, there are not many devices exhibiting this type of behaviour, therefore, these devices are removed from the data-set. Another example of a device that is removed from the data-set is a device that has very few flows associated to it (less than 5), since too few flows are not sufficient to create a relevant device profile.

The next modification that is done to the data-set is the extraction of domain names from the URLs. This step is necessary, since the dimensionality of the URLs is very high and essentially each URL is unique by itself due to its structure. The structure of the URL can be seen in Figure 5.2. The URLs also often contain tokens that are only used for one session, usually located in the query parameters, which would be different in the following session even for the same activity on the given website. To reflect this fact, the domain needs to be extracted, which extracts the informational value from the feature. Therefore, the extracted value from the example in Figure 5.2 is `domain-name.example-domain.com`. Some information contained within the URL, such as the port, is also an important feature, however, the port number is represented as a separate column in the data-set, so it does not need to be extracted, since it is redundant information.

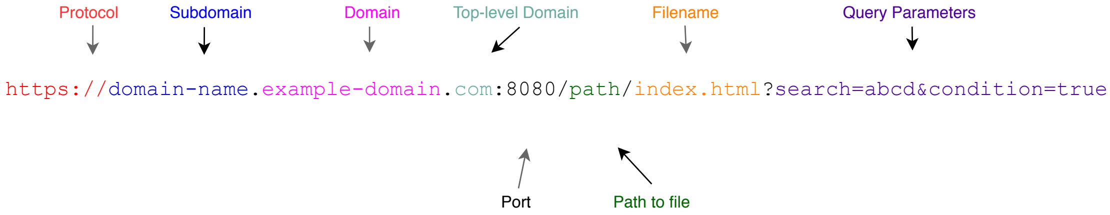

Protocol
Subdomain
Domain
Top-level Domain
Filename
Query Parameters
https://domain-name.example-domain.com:8080/path/index.html?search=abcd&condition=true
Port
Path to file

Figure 5.2: Illustrative structure of a URL.

After the aforementioned steps are applied to the data, the data-set is separated into multiple smaller sets with various compositions of available data fields, and sizes to evaluate the accuracy of the proposed methods.

5.3 Application Design

This section contains a detailed description of the internal structure of the tool for identifying network devices based on their behaviour. The goal of this thesis is to explore the efficiency of classification algorithms on a wide variety of data, since some of the fields are not always available. Therefore, the application is designed as a tool that is highly configurable and modular to facilitate any additions of methods in the future.

As mentioned before, the tool is highly modular and can be separated into three distinct parts. The first is the statistical module that provides insight into the data-set used in the experiments. For further details regarding this module see Section 5.1. The second part is the module encapsulating all of the proposed methods and their training functions, if the method requires it. Specifics regarding each of the proposed methods are discussed in the Chapters 3 and 4. The final part is a module that provides validation functionality to the tool and evaluates the accuracy of the Naive Bayes and Decision Tree classification methods. The class diagram describing the internal structure of the tool can be found in the Appendix D.

Since this tool is able to handle multiple device identification methods, it is necessary to provide information to the tool regarding the configuration of the methods, and also the meanings of the columns, since the data does not have a given structure and columns might need to be added in the future. Lastly, the configuration file contains an array of experiments that are going to be performed with different columns, since not all columns are always available and might worsen the predicted results. YAML has been chosen as the file format describing the parameters for the set of experiments due to legibility and the ease of parsing. The configuration files that are used to test the proposed methods are available on the attached DVD, the contents of which are described in Appendix E. An example configuration file is in Figure 5.3. The following paragraphs contain specifics regarding each of the proposed methods:

Naive Bayes Classifier – This classifier can be operated in only one mode, where it re-identifies devices that have been seen during the training phase. However, the disadvantage is that devices that did not appear in the training data-set are classified as one of the known devices, which is incorrect.

Decision Tree Classifier – The decision tree classifier requires the entire data-set to be within the memory, thus the limit of the size of the training data-set needs to be set according to the size of the installed RAM, the rest of the set will be ignored due to spacial constraints. Configurable parameters are following: the criterion based on which splits are measured and the maximum depth of the decision tree. This classifier identifies already seen devices, therefore, it is identifies new devices incorrectly. A solution to this problem would be to retrain the classifier after a period of time has passed. The length of the time-frame would be determined based on the frequency at which new devices appear in the data-set.

k-Nearest Neighbors Classifier – This classifier is very simple, however, it has the same limitations with memory size as the decision tree classifier. The only configurable value for this classifier is the variable k and the distance metric as described in Section 3.2. However, this method is very flexible in the way the data can be presented to it. This thesis proposes multiple approaches to flow aggregation described in Sections 6.3 and 6.4.

5.4 Technical Implementation

This section describes the technical implementation and the libraries that are used to create this tool. Furthermore, the benefits of the configurability of the tool are discussed within this section.

Python 3.7 has been chosen as the programming language for the implementation of this tool due to its simplicity and wide availability of machine learning libraries. It is a dynamic interpreted language that provides automatic garbage collection. Another advantage of using this language is the cross-platform compatibility between Windows, Mac OS and Linux.

```
columns:
  timestamp: 0
  companyID: 1
  MACaddress: 2
  srcIP: 3
  dstIP: 5
  url: 4
  port: 6
  TLSfingerprint: 7
  HTTPfingerprint: 8
delimiter: '\t'
createStats: true
classify: true
classificationMethod: NaiveBayes
classifierConfig: empty
label: MACaddress
experiments: [[companyID, srcIP, dstIP, port, url]]
```

Figure 5.3: Example of a configuration file for the Naive Bayes Classifier.

The proposed tool is using multiple libraries that provide a wide range of functionality such as machine learning classifiers or functions to create plots. The following libraries are going to be used in the implementation:

NumPy – NumPy is a community project that is specifically designed for scientific computing, that is nowadays essential in the field of machine learning [41]. It is used as an efficient high dimensional container for vectors of various types throughout this work.

Scikit Learn – This module is an open source project that provides a variety of supervised and unsupervised machine learning algorithms. It provides a standardized API which is easy to use and takes into account that data-sets can be larger than the installed memory in the computer. The library is built on top *scipy* and *numpy* [9].

PyOD – PyOD is a recognized toolkit in the machine learning community designed for outlier detection. This library contains a large variety of anomaly detection methods such as outlier detection using Neural Networks, Principal Component Analysis (shortly PCA), Isolation Forests and many others [47]. This work uses the k-NN outlier detection algorithm from this library described in Section 3.5.2.

Pandas – Pandas is also an open source project that aims to aid in data analysis and modeling. It specializes in data structures and operations for manipulating data tables [29].

Matplotlib – Matplotlib is a plotting library that provides the ability to create histograms, scatter plots and bar charts. This library is going to be used throughout the statistical and for creation of all of the graphs within this thesis.

Chapter 6

Experimental Results

This chapter contains a detailed description of the experiments performed, their settings and achieved results. The goal of this chapter is to find the optimal settings of the device identification methods and evaluate their overall accuracy on the tested data-sets. A detailed description of the data-sets used in the experiments is located in Appendix A. Experiments are focused on accuracy and not on speed, since there is a lot of overhead introduced into the tool, that in return enables easy creation of experiments, setting different parameters, and creation of statistics, while hindering the classification speeds of the algorithms.

This chapter of the thesis can be separated into two parts. The first part contains the evaluation of algorithms in Sections 6.1 and 6.2 that would need to be retrained with any addition of new devices. The second part evaluates two new approaches in Sections 6.3 and 6.4 to methods that are able to continuously update and track network devices. From the aforementioned algorithms, the latter method is the most successful from all of the tested methods. Therefore, this method is compared in Section 6.5 with the already existing tracking algorithms and analyzed in regard of accuracy and their limitations.

The initial step in each of the classification methods is to evaluate their accuracy with a varying number of available data fields (features). This step is necessary to rule out features that reduce the accuracy of the model. Since these experiments are going to be the same for all of the proposed methods, a standardized approach towards testing can be taken. Therefore, Table 6.1 provides details about the initial experiments, and the names for the experiments are used in all of the following sections. The rest of the experiments are varying parameters that are specific to the classifier, therefore, are evaluated individually with each of the device identification methods.

Experiment Name	Features Used
feature_exp1	srcIP, dstIP, port, TLSfingerprint, HTTPfingerprint, domain
feature_exp2	srcIP, dstIP, port, TLSfingerprint, HTTPfingerprint
feature_exp3	srcIP, dstIP, port, HTTPfingerprint, domain
feature_exp4	srcIP, dstIP, port, TLSfingerprint
feature_exp5	srcIP, dstIP, port, domain

Table 6.1: Table containing the naming scheme for the initial experiments varying the used features for the classifier that is identical for all of the methods in this thesis.

6.1 Naive Bayes Classifier

The Naive Bayes Classifier (see description in Section 3.3) is a very simplistic classification method that does not have many options in fine tuning the parameters of the model. The only parameters that can be varied are the features that are used in the classification and the distribution. Since there are multiple devices in the data-set, only the `multinomial` and `Gaussian` distributions can be used. Tables 6.2 and 6.3 describe the achieved results of the Multinomial Naive Bayes Classifier and the Gaussian Naive Bayes Classifier with varying features respectively. These methods are tested using `Data-set 1` and validated using k-Fold cross-validation algorithm with *k=5*.

Experiment	Accuracy	Precision	Recall	F1
feature_exp1	0.05215	0.13331	0.08349	0.09426
feature_exp2	0.05184	0.13308	0.08406	0.09430
feature_exp3	0.05477	0.13091	0.08325	0.09331
feature_exp4	0.05192	0.13345	0.08341	0.09425

Table 6.2: Experimental results with varying features (description in Table 6.1) using the Naive Bayes Classifier using the multinomial distribution. Best result is marked in green.

The best accuracy achieved in this set of experiments with the multinomial distribution is 5.477% in the experiment `feature_exp3`. This is due to the fact that the classifier does not take any of the features into consideration, and merely categorizes the device based on the distribution of data within the training data-set [43]. Thus, this type of classifier is not efficient enough to be considered as a solution, but merely as the simplest model baseline.

Experiment	Accuracy	Precision	Recall	F1
feature_exp1	0.55300	0.65513	0.68471	0.60043
feature_exp2	0.55954	0.65631	0.68785	0.60395
feature_exp3	0.65855	0.73311	0.83833	0.74111
feature_exp4	0.55998	0.65917	0.69457	0.60737

Table 6.3: Experimental results with varying features (see Table 6.1) using the Naive Bayes Classifier using the Gaussian distribution. Best result is marked green.

In the second set of experiments, similarly to the first set, the best accuracy of 65.855% is achieved in the experiment `feature_exp3`. The improvement in accuracy is attributed to the fact, that the model employs the Gaussian distribution. This draws the speculation, that the data has a distribution that is close to the normal curve, since this model achieves better results than the aforementioned multinomial distribution. However, the achieved accuracy is not satisfactory for the production environment, and further methods need to be explored.

To summarize, neither of the Naive Bayes classifiers perform to a degree that would be considered a successful method for network device identification. Thus, other device identification methods are explored in the following sections.

6.2 Decision Tree Classifier

Decision Trees provide a simple and efficient representation for classification. Similarly as the Naive Bayes Classifier, it does not have many parameters to vary in order to find the model with the best performance. Parameters that can be varied include the metrics based on which splits are made and the depth of the tree. This section describes all of the experiments performed using the Decision Tree Classifier. Depths of the trees are not tested, since these are dependent on the data-sets and the number of devices within them. Therefore, only the split metrics are tested. The tested metrics include the Gini impurity and entropy, and their performance is located in Tables 6.4 and 6.5 respectively. All of the tests using this classifier are performed on the `Data-set 1`. The results are cross-validated with $k = 5$, and the results in the tables contain the mean values from all of the iterations.

Accuracy	**Precision**	**Recall**	**F1**
0.96963	0.93392	0.93454	0.93261

Table 6.4: Experimental results using the Descision Tree Classifier with the Gini impurity quality of the split measure.

Accuracy	**Precision**	**Recall**	**F1**
0.96964	0.93400	0.93439	0.93252

Table 6.5: Experimental results using the Decision Tree Classifier with the Entropy quality of the split measure.

Based on the results (see Tables 6.4 and 6.5), the decision tree classifier achieves a high prediction accuracy using both metrics. The difference in accuracy between the two metrics is statistically insignificant (only 0.001%), therefore, both metrics are considered to provide accurate models for the tested data-set.

On the other hand, this classifier, similarly as the Naive Bayes Classifier, would need to be retrained with any newly appearing devices. Another disadvantage of this classifier is that the splits are made based on the feature that provides the best accuracy, and those features, such as the source IP address, might change over time, which will yield a lower prediction accuracy in the future. A solution to this challenge would be to retrain the model after a certain period of time, which requires time and memory resources. The time between retraining the model might also vary based on the variations in user behaviour, and in cases of an unexpected problem within a company, where users search for solutions, the model would need to be retrained more often than the scheduled re-training time.

Based on the aforementioned disadvantages, this model could only be used in a scenario where the devices have statically assigned IP addresses and their user behaviours do not vary much over time, which is true in a factory context. On the other hand, in cases where IP address are assigned statically, they also serve as a unique identifier of a device, therefore, device tracking algorithms are not necessary. However, the goal of this thesis is to find a universal algorithm that would be able to track devices in any type of network varying from a private network, up to a large university or corporate network. Therefore, other methods for device identification need to be evaluated, which could be used without any limitations to the network architecture.

6.3 Nearest Neighbor Combined Model

The Nearest Neighbor Combined Model, described in the Section 4.1, is model inspired by text similarity methods. It features a simple approach to identifying devices by searching for the closest neighbor to the currently classified device profile. As with the previous approaches to network tracking, the first step during the evaluation phase is the process to determine features that achieve the highest accuracy on the tested `Data-set 2` (see description in Appendix A). The first hour of the exported data is used to create the initial profiles for the devices. Profiles are aggregated based on the combination of the MAC address and username. Classification is performed on the following hour, where the timeline is segmented into 5-minute time windows, with the classification profiles being aggregated based on the source IP address and username. The Table 6.6 summarizes the achieved accuracy with varying feature sets. Based on the results from the table it has been concluded that all of the features are necessary to achieve the best results even in the cases where the value of the feature is sparsely available.

Experiment	**Accuracy**
feature_exp1	0.94580
feature_exp2	0.87164
feature_exp3	0.83752
feature_exp4	0.08327
feature_exp5	0.00001

Table 6.6: Experimental results with varying features (description in Table 6.1) using the Nearest Neighbor Combined Model. The best performing combination of features is highlighted green.

After the combination of features for best performance is found, it is necessary to evaluate the behaviour of profiles in a high dimensional space. Therefore, the t-SNE algorithm to reduce dimensionality is used to visualize the profiles created during the training phase. Visual output from the dimensionality reduction algorithm can be seen in the Figure 6.1. The graph shows that most of the device profiles do not create clusters and have a distinct space between one another. Since the profiles are separated, the devices that are being classified will have a lower probability of being mistaken for another device, which in return yields a higher accuracy.

One of the major challenges when analyzing vectors with high dimensionality is choosing the correct distance metric. Among the most commonly used similarity metrics in text similarity analysis is the cosine similarity. When performing similarity measures between profiles, it is expected that devices with the same and different MAC addresses are represented by two roughly disjoint normal distributions. The following Figure 6.2 represents the distributions of similarities between profiles. The mean and standard deviations for the same MAC addresses have the values of $\mu = 0.8901$ and $\sigma = 0.0363$ respectively. The similarity values for profiles with different MAC addresses follow the normal distribution as well, with the mean $\mu = 0.7243$ and standard deviation $\sigma = 0.0497$. These distributions are not disjoint, however, achieve a high performance in distinguishing and identifying network devices.

The last experiment performed using this algorithm is the analysis of its performance over a longer time period. As in the previous experiment, the first hour of the exported data is used to create the initial profiles and the following 8 hours are used for the evaluation

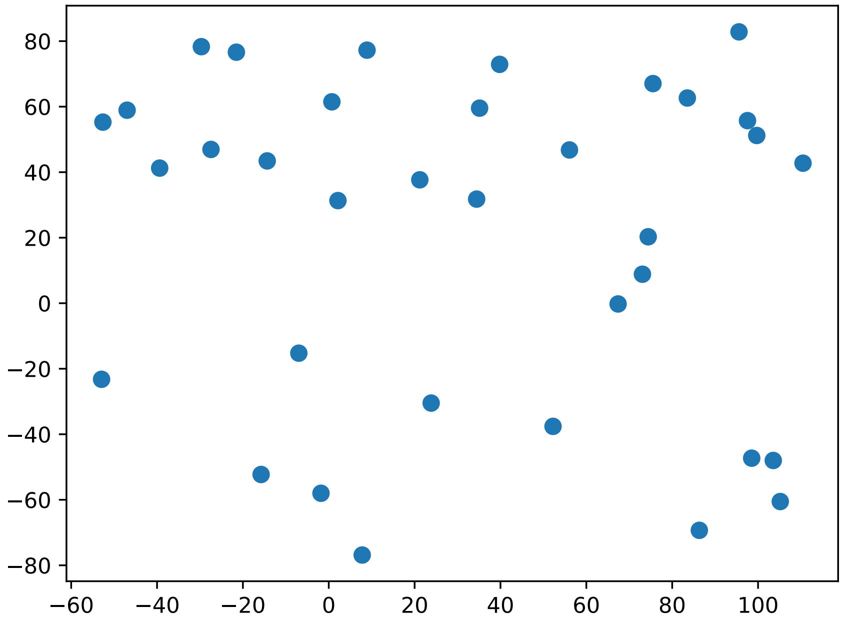


Figure 6.1: Graphical output obtained using the t-SNE algorithm to reduce dimensionality. Each one of the data points represents one of the profiles that are used in the Nearest Neighbor Combined Model.

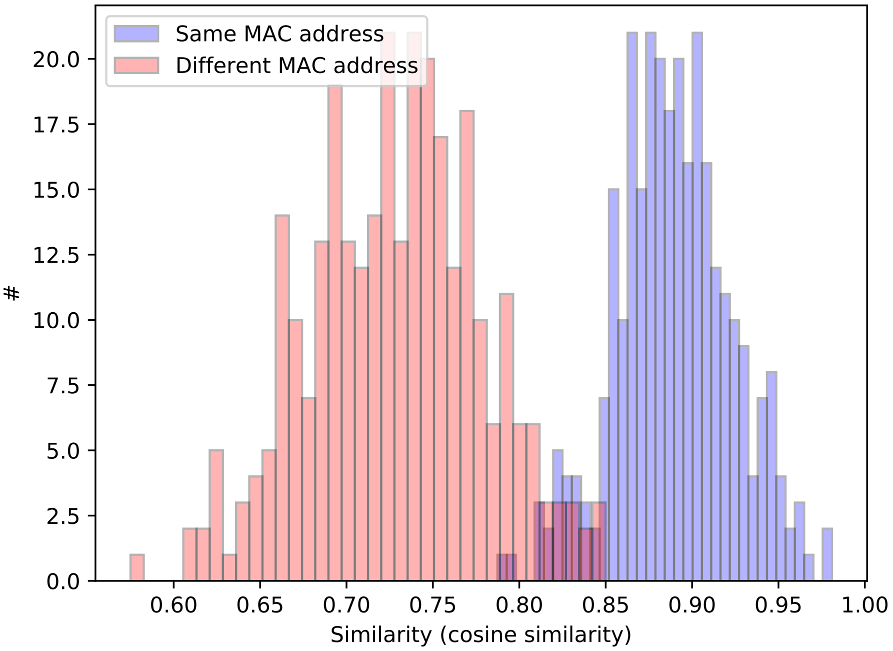


Figure 6.2: Histogram illustrating the distribution of similarity values (cosine similarity) for devices with the same MAC addresses (in blue) and different MAC addresses (in red) in the Nearest Neighbor Combined Model. This graph shows that the distribution of similarities for the devices with the same and different MAC addresses roughly follow the normal distribution.

of the performance of the algorithm. The Table 6.7 contains the achieved results during the tested time-period. The algorithm achieves an average tracking accuracy of 82.652% over the tested time-period. However, the precision, recall and F1 score have a low value, which indicates that the algorithm classifies many devices as false positives or false negatives. Low values of these scores and accompanying incorrect classifications make these models for devices unsuitable for a production environment. Therefore, a modification to this approach has been proposed in the Section 4.2, and subsequently tested. The performance of the modified algorithm is described in the Section 6.4.

Hour	**Accuracy**	**Precision**	**Recall**	**F1**
2	0.94580	0.42925	0.43485	0.43208
3	0.90306	0.29024	0.29417	0.29131
4	0.87882	0.24661	0.25009	0.25002
5	0.84952	0.24536	0.24917	0.24949
6	0.80218	0.24717	0.25088	0.25053
7	0.76183	0.21728	0.23205	0.22611
8	0.74765	0.20992	0.21289	0.21135
9	0.72384	0.19367	0.19614	0.19483

Table 6.7: Experimental results for the algorithm over time using the Nearest Neighbor Combined Model.

6.4 k-NN with Segmented Profiles Model

In the initial stages during the design of this algorithm it is necessary to evaluate the behaviour of profiles in a high dimensional space in order to find methods that efficiently identify devices. Therefore, similarly as in the previous section, the t-SNE algorithm to reduce dimensionality is applied to the profiles. Profiles are created by aggregating flows in a 5-minute time-window based on their source IP address and username. The visual output of the algorithm is located in the Figure 6.3. This graph illustrates that almost all of the device profiles have a tendency to create clusters that are disjoint. The fact that the device profiles have a tendency to create clusters is a beneficial property as k-NN tends to perform better under these conditions [28].

The performance of k-NN in regard to time complexity is high $\mathcal{O}(n^2)$, therefore, classifying network devices assigned to only one username is a beneficial reduction time instead of testing all of the device profiles stored in the memory [10]. This reduction in time is significant since the maximum of profile vectors assigned to a username is 51 in the tested data-sets. Furthermore, this form of clustering can be used for outlier detection.

Since device profiles create clusters that are disjoint, new devices will have a tendency to lie elsewhere in the high dimensional space. Therefore, methods for outlier detection that are introduced in Sections 3.5.1 and 3.5.2 are used for new device detection. The effect on accuracy of these methods is explored further in this section.

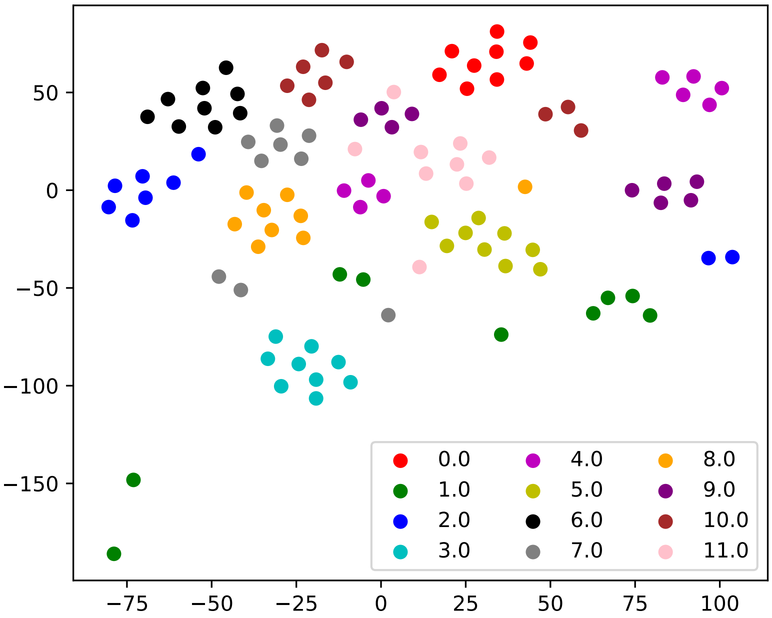


Figure 6.3: Graphical output obtained using the t-SNE algorithm to reduce dimensionality. The numbered labels and colors represent devices and data points represent device profiles. This graph illustrates how almost all of the device profiles create disjoint clusters.

The following step after the analysis of the device profile behavior is to determine an appropriate distance metric for the k-NN algorithm. In order for the metric to achieve the desired result, it is necessary for the similarities of device profiles with the same and different MAC addresses to follow two separate, ideally disjoint, distributions. The following Figure 6.4 describes the distribution of values between the same and different devices. The device pairs that are used for the comparison of similarity have been chosen at random to get the best representation of the distribution. The histogram draws the conclusion that the distribution of similarities for devices with the same MAC address follows roughly the normal distribution with the mean $\mu = 0.978$ and standard deviation of $\sigma = 0.0199$. Since the similarity values follow the normal distribution it is possible to use the z-score outlier detection described in Section 3.5.1 to detect new devices in the data-set. Therefore, devices that have all of the similarity scores of the closest neighbors further than three standard deviations from the mean $\mu = 0.978$ of the same device similarity, are considered new previously unseen devices.

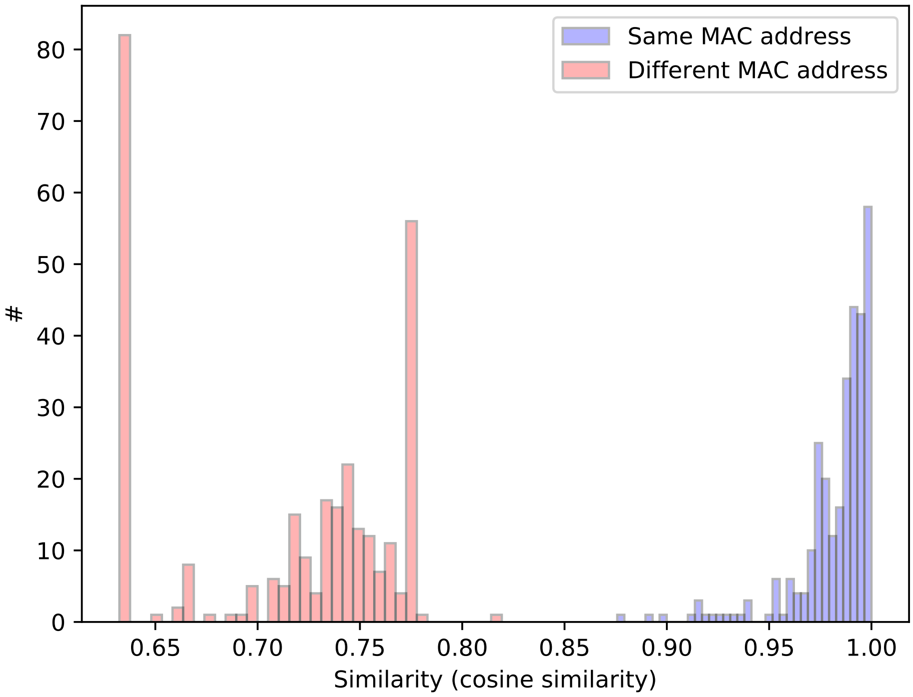


Figure 6.4: Histogram illustrating the distribution of similarity values for devices with the same MAC addresses (in blue) and different MAC addresses (in red). This graph shows that the distribution of similarities for the devices with the same MAC address roughly follows the normal distribution.

All of the experiments that are described in this section are performed on `Data-set 2`. Due to the nature of the tested `Data-set 2` with many missing values, it is first necessary to determine how do the available or unavailable features influence the accuracy of the proposed algorithm. Therefore, the first experiment evaluates the accuracy of the algorithm where the first hour is used for profile creation based on the MAC address and username in 5-minute time-windows. The second hour is used for scoring the accuracy without the detection of new devices or profile updates, where k-NN is used with k equal to 3. Similarly to the training phase, a 5-minute time-window is used and the profiles are aggregated based on their source IP address and username. Subsequently, the MAC addresses are used as the labels to evaluate the accuracy of the prediction. The following Table 6.8 describes the accuracy of the algorithm. The experiment with the features `feature_exp1` achieves the highest accuracy and it is concluded that even features that are often not available, such as TLS or HTTP fingerprints, improve the accuracy of the proposed method. Therefore, from this point on-wards in this section the most successful combination of features `feature_exp1` is used in all of the experiments.

Experiment	Accuracy	Precision	Recall	F1
feature_exp1	0.93562	0.89324	0.93020	0.89503
feature_exp2	0.93610	0.87732	0.91992	0.89811
feature_exp3	0.92905	0.86941	0.89356	0.88132
feature_exp4	0.13229	0.10482	0.08649	0.09473
feature_exp5	0.00001	0.00001	0.00001	0.00001

Table 6.8: Experimental results with varying features (description in Table 6.1) using the k-NN algorithm with segmented profiles.

6.4.1 Evaluation of the Impact of k Using the k-NN Algorithm

After the evaluation of accuracy using various combination of features, it is necessary to determine the optimal value for k for the k-NN algorithm. Therefore, the following experiment is performed, where the first hour is used for training the model and the following hour is used for accuracy evaluation. Profiles are aggregated using the same strategy as in the previous experiment based on the username and MAC address during the training phase, and username and source IP address during the classification phase. In this experiment the sliding window or the outlier detection are not used. The Table 6.9 contains the achieved results.

The achieved results signify that the value $k = 3$ achieves the highest accuracy. This is true due to the fact that the lower value of k enables the classifier to identify devices that have not communicated often during the training phase. Since the low k achieves the best results, this enables the algorithm to classify new devices correctly earlier than the higher values of k, due to the fact that a lower k requires fewer created profiles to classify the current profile as said device. This fact is then further exploited in the section describing new device detection.

k	Accuracy	Precision	Recall	F1
3	0.93562	0.89324	0.93020	0.89503
5	0.92588	0.87880	0.91502	0.87991
7	0.92131	0.87709	0.90421	0.87772
9	0.89471	0.83954	0.87209	0.83830
11	0.85238	0.79540	0.82673	0.79345

Table 6.9: Experimental results for various values for k in the k-NN algorithm. The experiment that has achieved the best results is highlighted green.

6.4.2 Evaluation of the Impact of the Time-Window Size

With the optimal k along with the set of features decided, it is necessary to find the best performing size of the time-window used for segmentation. The window needs to be short enough to minimize the probability of aggregating multiple devices into one profile, however, the length also needs to be as long as possible to minimize the number of classifications within one hour. In order to determine the optimal length in seconds, multiple experiments are performed evaluating the accuracy of the proposed algorithm. The following Table 6.10 summarizes the results of the experiments varying the size of the time-window. The profiles have been aggregated using the same method as described in the previous experiments based on the username and source IP address, with $k = 3$ without the use of the sliding window to update profiles nor algorithms for new device detection. Based on the experimental results, it has been concluded that the best performing time-window length of segmentation is equal to 5 minutes (e.g., 300 seconds). This time-window is also short enough to reflect short DHCP leases that are used in networks with a lot of movement of users.

Window Length [s]	**Accuracy**	**Precision**	**Recall**	**F1**
180	0.90179	0.85646	0.88769	0.85768
300	0.93733	0.89287	0.92646	0.89529
480	0.93562	0.89324	0.93020	0.89503
600	0.93279	0.88289	0.92130	0.88614

Table 6.10: Experimental results for various lengths of the time window using the k-NN algorithm. The experiment that has achieved the best results is highlighted green.

6.4.3 Evaluation of the Impact of Stored Profiles

When the optimal value of k and the length of the time-window have been determined, it is required to find the optimal number of profiles stored in the memory. Since the number of devices contained within a large corporate network can be pretty high (in excess of 10,000 devices), the algorithm requires the number of stored profiles to be as low as possible, since memory is a limited resource. However, the trade-off for lower memory requirements also hinders the accuracy of new device detection that is used further in this section. Therefore, a compromise needs to be done between spatial complexity and the ability to detect new devices, since algorithms such as k-NN perform better in circumstances where more data points are available. The following Table 6.11 evaluates accuracy with varying number of profiles stored in the memory. The methodology behind this experiment is the same as in the previous experiments, the length of the time window is 5 minutes, and $k = 3$ without profile updates. Neither of the algorithms for profile updates nor new device detection are used.

Profiles	**Size [MB]**	**Accuracy**	**Precision**	**Recall**	**F1**
6	1937	0.93544	0.89308	0.92999	0.89486
8	2357	0.93562	0.89319	0.93021	0.89499
10	2986	0.93562	0.89322	0.93022	0.89501
12	3242	0.93560	0.89321	0.93020	0.89503

Table 6.11: Experimental results with varying number of profiles per device stored in the memory and their spatial complexity.

Based on the results from the Table 6.11, the number of stored profiles does not have a significant influence on the accuracy of the proposed algorithm. From further exploration of the results, this was caused by the higher similarities of the newer profiles (e.g., the last 2 or 3), and it does not seem to be important to store a higher number of profiles, however, updating the profiles with newer ones during the classification phase will have a positive impact on the accuracy of the algorithm over time. Even though the number of profiles does not have a significant influence on the accuracy of the algorithm, outlier detection will benefit from the availability of more profiles. Therefore, from this point on-wards the algorithm stores 12 profiles in the memory that are going to be available later in this thesis and used for outlier detection. Another reason why more profiles in the memory are not a big issue is that the increase in memory complexity is not linear as it might be expected. This is caused by devices not communicating every 5-minute time window, which results in less profiles created for said device compared to other devices that communicate more often over time.

6.4.4 Evaluation of the Impact of the Sliding Window

As it has already been mentioned in the previous section, due to the higher similarities in newer profiles of devices, the algorithm benefits from regular updates over time. This is caused by the change of behaviour of users and devices over time, therefore, the previously created profiles need to be updated and the older profiles need to be replaced by newer ones.

Device users tend to change the context of their work approximately every hour, therefore, the number of 5-minute profiles that are stored in the memory is limited to 12. If there are 12 profiles, each time a device is classified as a specific device, the oldest profile is deleted and the current one is added into the profile pool. In the case that there are less than 12 profiles, the newest profile is directly added to the profile pool. The Table 6.12 summarizes the achieved accuracy over time without the use of the sliding window method. Afterwards, the following Table 6.13 contains the performance of the algorithm using the sliding window for comparison. The Figure 6.5 compares the accuracy achieved over the tested time period of 8 hours.

Hour	Accuracy	Precision	Recall	F1
2	0.91304	0.88956	0.92931	0.89126
3	0.87210	0.84849	0.90754	0.84961
4	0.85226	0.80795	0.88912	0.80919
5	0.84592	0.77872	0.87426	0.78037
6	0.82762	0.75254	0.86017	0.75481
7	0.80510	0.73703	0.85142	0.73946
8	0.78574	0.72752	0.84426	0.72963
9	0.75068	0.72366	0.84124	0.72577

Table 6.12: Experimental results representing the accuracy over time of the base algorithm without the sliding window.

Based on the aforementioned Tables 6.12 and 6.13, a conclusion is drawn that the sliding window improves the accuracy significantly over time. This improvement is mainly seen in the last hour (hour 9) of the classification, where the extended algorithm represents an improvement of over 8% in accuracy compared to the base algorithm without the use of the sliding window. Therefore, in conclusion, the sliding window improves the overall accuracy of the proposed algorithm in the worst case scenario, where no labels are present in the classification phase.

6.4.5 New Device Detection

Previous experiments do not identify newly appearing devices and are only capable of re-identifying devices learned during the training phase. Therefore, algorithms such as *z-score outlier detection* and *k-NN outlier detection* are introduced to counteract this challenge,

Hour	Accuracy	Precision	Recall	F1
2	0.92855	0.89815	0.93714	0.90031
3	0.90696	0.86116	0.91963	0.86295
4	0.88689	0.81799	0.89923	0.82030
5	0.87956	0.79107	0.88519	0.79327
6	0.86123	0.76723	0.87079	0.76893
7	0.84820	0.75270	0.86247	0.75451
8	0.83831	0.74328	0.85536	0.74492
9	0.83319	0.74001	0.85298	0.74167

Table 6.13: Experimental results representing the accuracy over time of the base algorithm with the sliding window.

since devices have a tendency to appear and disappear due to movement of users, irregularly scheduled meetings, lectures, etc.

The first tested outlier detection algorithm is the *z-score outlier detection* due to its simplicity and ease of implementation. Since the distribution of profile similarities for the same MAC addresses roughly follows the normal distribution, as it is seen in Figure 6.4, it is possible to apply the empirical 68-95-99.7% rule as described in Section 3.5.1 to detect new devices. The distribution for the similarities of devices with the same MAC addresses has the mean $\mu = 0.978$ and standard deviation of $\sigma = 0.0199$. Based on these values any device whose similarities are below the threshold 0.9183 should be considered outliers or in the case of this thesis new devices. The following Table 6.14 contains the experimental results for the threshold 0.9183. Devices which have all similarities obtained from the k-NN algorithm below this threshold are considered new, previously unseen devices. Devices that have exactly one similarity above this threshold are classified as said device. With multiple device similarities above this threshold the k-NN algorithm is applied.

In this experiment, the algorithm is trained on the first hour of the exported data, and is run on the following 8 hours of the data-set. The algorithm is using the sliding window to update the device profiles. The summary of the settings of this experiment can be seen in the Appendix B.

Hour	Accuracy	Precision	Recall	F1
2	0.93363	0.88963	0.92914	0.89127
3	0.91280	0.84907	0.90806	0.85019
4	0.89284	0.80849	0.88994	0.80995
5	0.87682	0.77940	0.87509	0.78123
6	0.85854	0.75320	0.86085	0.75569
7	0.84619	0.73784	0.85210	0.74048
8	0.83695	0.72849	0.84509	0.73089
9	0.83199	0.72843	0.84221	0.72726

Table 6.14: Experimental results for the base algorithm using the sliding window and z-score outlier detection over time.

The results in the Table 6.14 indicate that the accuracy is improved in the initial hours of the classification (hour 2 through 4) compared to the base algorithm with the sliding window (see Table 6.13) by approximately 2%, however, in the later hours of the export,

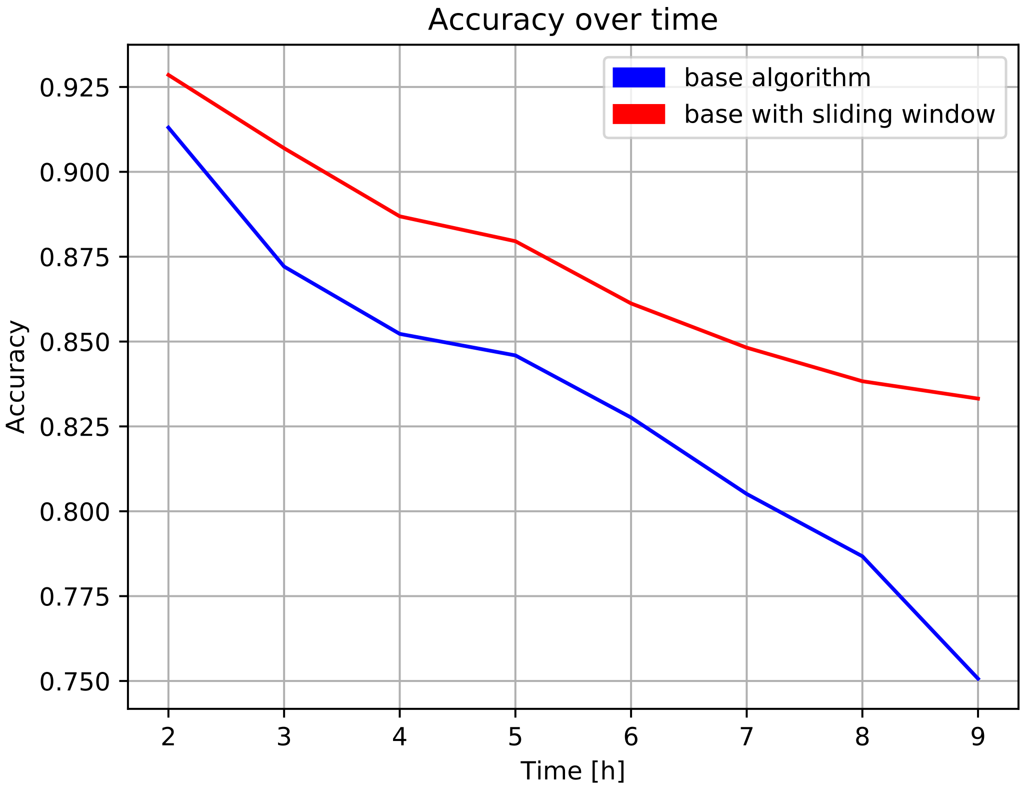


Figure 6.5: Comparison of the accuracy over time of the base algorithm vs. the modification with the sliding window to update device profiles.

the accuracy slightly deteriorates by about 0.2%. This decrease is insignificant and will not play a role in a real world application of this algorithm since it represents a worst case scenario, where none of the labels are available after the training phase. The effect will be mitigated by the occasional availability of labels, which would cause a correct update of device profiles, therefore, achieving results that are in the neighborhood of the results achieved in hour 2 or 3.

Lastly, the *k-NN outlier detection* algorithm is tested using the library *PyOD*. Outlier detection algorithms implemented in this library require the $k = 5$ to achieve the best results. Therefore, new device detection using this method is performed only on device profiles assigned to a username that has at least 5 profiles overall, which can span across multiple devices. However, this approach does not take into the account the devices that have only one profile created for them after being classified as a new device. Therefore, similarly to the previous outlier detection algorithm, devices whose exactly one distance obtained from the k-NN algorithm is above the aforementioned threshold of 0.9183 are classified as said device. In the rest of the cases the k-NN algorithm is used to decide whether the device is new or not.

The first hour of the export is used for training, profiles are aggregated based on their username and MAC address during the training phase, afterwards based on their source IP address and username, with 5-minute time-windows, $k = 3$, with the sliding window and the following 8 hours are classified. Table 6.15 summarizes the achieved results in the last hour of the classification e.g., hour 9 with varying distance metrics to detect previously unseen devices.

Distance Metric	Accuracy	Precision	Recall	F1
Minkowski distance	0.81945	0.71152	0.84222	0.72032
Dice coefficient	0.81979	0.71132	0.84173	0.72007
Cosine similarity	0.81911	0.72025	0.81036	0.71265

Table 6.15: Results of the experiment with varying distance metrics used for the k-NN outlier detection algorithm. Results in this table are cumulative for the last hour of the classified time period. The best performing metric for detecting outliers is highlighted green.

Experimental results from the Table 6.15 show that the distance metrics do not have a significant impact on the accuracy of new device detection, however, the Dice similarity coefficient achieves the best results by an insignificant margin. Furthermore, the achieved accuracy is worse compared to either the *base algorithm with the sliding window* or the *z-score outlier detection with sliding window* algorithms. The graph in Figure 6.6 summarizes and compares the achieved results of the base algorithm with the sliding window, z-score outlier detection with sliding window and k-NN outlier detection with profile updates algorithms.

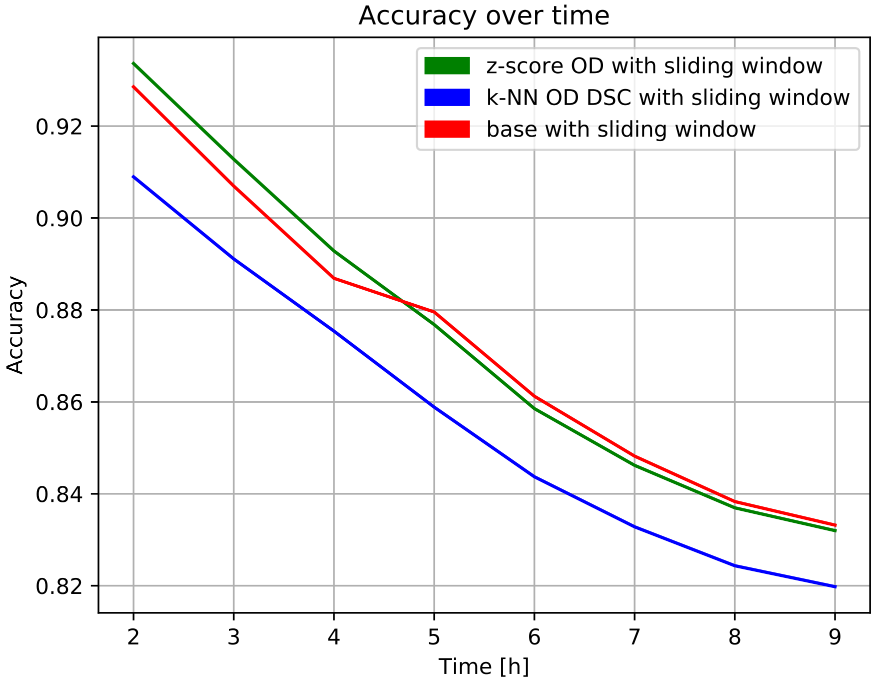


Figure 6.6: Final comparison of the achieved accuracy with varying modification to the segmented profiles using k-NN algorithm. This graph contains the initial version of the algorithm with the sliding window, then the base algorithm with the sliding window with z-score outlier detection, and finally, the k-NN outlier detection with Dice similarity coefficient to detect outliers with updates to the profiles.

To conclude, this section performed experiments varying the parameters of the algorithm proposed in Section 4.2. The experiments are performed in the worst case possible, where none of the labels are available during the classification phase, which rarely happens in a production environment. The best performing variant of the algorithm is the variant with the z-score outlier detection combined with the sliding window used to update device profiles. A summary of the settings of the best performing proposed algorithm is in Appendix B. This algorithm achieves the average accuracy of 89% over the tested 8 period and provides a reliable method of identifying and tracking devices without the knowledge of unique identifiers such as the MAC address.

6.5 Comparison with Existing Methods

The last section of this chapter serves as a comparison of the achieved performance of the proposed algorithm in Section 4.2 with already existing methods described in the Chapter 2. The methods are compared in regards to their accuracy, speed, universality of use, and their specific nuances compared to the proposed algorithms in this work.

The initial comparison is made with the clock-skew device identification method proposed by Kohno et al. described in the Section 2.5. This method relies on the uniqueness of the frequency of the crystal that is used to measure time in computers. Clock-skew provides a reliable method of identifying specific devices achieving the accuracy of 98% [45]. Therefore, this method could be used to identify the set of IPv6 addresses assigned to any given device in future research. Even though the proposed algorithm achieves a lower accuracy of 89%, it provides a passive approach to identification, and does not require additional data outside of the data generated by the users.

HTTP and TLS fingerprinting, similarly to the algorithms proposed in this work, represent passive device tracking techniques. However, the major disadvantage of the aforementioned mechanisms is that they require a large database of fingerprints to be stored for each device and website the devices visit [2]. Furthermore, it becomes a challenge to search among this database for the correct fingerprints, which consumes a significant amount of time. The last disadvantage of fingerprinting is the constant changes in these identifiers, which are a challenge to keep up to date in the database [13]. On the other hand, the HTTP fingerprinting method deems to be accurate in contexts such as technically apt users who have specific add-ons and fonts installed on their browsers, achieving the accuracy of 96.23% [27, 24]. However, in the context of general public users HTTP fingerprinting achieves the accuracy of only 65% [13]. This discrepancy in results does not provide a reliable method to identify network devices, however, very few devices share the same fingerprint, which can be used to reduce the set of possible devices significantly [24]. Therefore, HTTP and TLS fingerprinting are used in this thesis as one of the features to track devices based on their behaviour.

The last method compared to the algorithm designed in this thesis is proposed by Kumpost [20], who reduces the dimensionality of the source IP address feature, and afterwards uses cosine similarity to find the closest neighbor to the current vector. Kumpost's method achieves an accuracy of 78.3%, however, has high spatial and time demands over large data-sets, which renders this method impractical in an environment with many network devices. Another disadvantage of his approach is that the aforementioned method has only been tested on SSH traffic where all of the features are always available, which is rarely the case in a corporate environment with varying configurations of routers.

To conclude this section, the proposed k-NN with Segmented Profiles Model provides a universal and reliable technique to identify network devices by using a combination of the previously discussed algorithms, such as HTTP and TLS fingerprinting, and removing their disadvantages, such as their spatial complexity. It achieves an average accuracy of 89% over the tested eight hour time-period. One of the major advantages compared to the previous approaches is its universality and independence on a specific protocol. Furthermore, profile representation is memory efficient compared to TLS and HTTP fingerprinting methods. The proposed method of device identification is suitable for network administrators, who need to track the devices on the network when unique identifiers such as the MAC address are not available.

Chapter 7

Conclusion

The goal of this thesis is to design methods of identifying network devices based on their behaviour. Two traditional machine learning methods have been tested, and two algorithms have been proposed to identify and track network devices without the necessity to retrain them after each new device is added to the network.

This thesis can be divided into four parts. The first part in Chapter 2 is an introduction to networking concepts used throughout this thesis. Methods, along with research and their results that have been used in the past for device identification are described within this chapter as well. The following Chapter 3 describes machine learning algorithms such as the Naive Bayes Classifier, Decision Trees and methods used to measure similarity of text in great detail. The following Chapter 4 contains the description of two newly proposed algorithms for network device tracking and identification that do not require to be retrained with the addition of any new devices. Chapter 5 elaborates on the design of the tool for experimenting with device identification algorithms. Afterwards, these methods are thoroughly tested and tuned for best performance in Chapter 6.

The analysis revealed the strengths and weaknesses of the tested algorithms. These results have then been used to further improve the accuracy of the algorithms. Experiments are listed in the order in which they have been performed, to show how the algorithms improved with each of the modifications.

All of the proposed algorithms are aggregated into a tool that enables easy experiment creation and evaluation of accuracy and other metrics. The best performing algorithm for network device detection is one of the newly proposed algorithms, the k-NN with Segmented Profiles Model. This method achieves an overall average accuracy of 89% on the tested dataset with over 10,000 devices. Therefore it is concluded, that this algorithm provides an efficient and reliable method of tracking, identifying, and detecting newly appearing devices without the knowledge of unique identifiers such as the MAC address. Lastly, the k-NN with Segmented Profiles Model is compared to already existing device identification methods in Section 6.5. The intended use of this algorithm is to provide network administrators the ability to automatically identify devices based on their behaviour without the necessity of features being always available.

Possible improvements to the proposed algorithms include parallelization, and better forms of outlier (new device) detection. Parallelization is a necessary improvement in the cases of large networks with a lot of movement of users to improve the performance of the algorithm. More outlier detection algorithms also need to be tested, since the improvement compared to the technique without new device detection is only 2% and could still be enhanced, due to the fact that most of the incorrect classifications are devices with no profiles created for them. Further research in this field could be performed by adding methods such as clock-skew, which could be used to determine the set of IPv6 addresses assigned to a given device, therefore, eliminating multiple profiles created for one device, since profiles are aggregated based on their source IP address and username [34].

An article about the proposed algorithm has been published as a part of the Excel@FIT 2020 conference. The poster in the Appendix C has been used to present the paper during the participant exhibition of their work.

Bibliography

[1] Aggarwal, C. C., Hinneburg, A. and Keim, D. A. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In: Bussche, J. Van den and Vianu, V., ed. *Database Theory — ICDT 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, p. 420–434. ISBN 978-3-540-44503-6.

[2] Anderson, B. *TLS Fingerprinting in the Real World*. April 2019. Online; visited 12.12. 2019. Available at: https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world.

[3] Anderson, B., McGrew, D. and Kendler, A. *Classifying Encrypted Traffic with TLS-Aware Telemetry*. January 2016. Online; visited 12.12. 2019. Available at: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=449962.

[4] B. Claise, E. *Cisco Systems NetFlow Services Export Version 9* [Internet Requests for Comments]. RFC 3954. RFC Editor, 2004. Available at: https://www.rfc-editor.org/rfc/rfc3954.txt.

[5] Bishop, C. M. *Pattern Recognition and Machine Learning*. 1st ed. Springer, 2006. ISBN 978-0387-31073-2.

[6] Brotherson, L. *TLS Fingerprinting*. February 2016. Online; visited 12.12. 2019. Available at: https://github.com/synackpse/tls-fingerprinting.

[7] Brownlee, J. *A Gentle Introduction to k-fold Cross-Validation*. May 2018. Online; visited 16.11. 2019. Available at: https://machinelearningmastery.com/k-fold-cross-validation/.

[8] Brownlee, J. *A Gentle Introduction to the Bootstrap Method*. May 2018. Online; visited 23.12. 2019. Available at: https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/.

[9] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A. et al. API design for machine learning software: experiences from the scikit-learn project. *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, p. 108–122.

[10] Cai, Y. and Wang, X. The Analysis and Optimization of KNN Algorithm Space-Time Efficiency for Chinese Text Categorization. In: Lin, S. and Huang, X., ed. *Advances in Computer Science, Environment, Ecoinformatics, and Education*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 542–550.

[11] Drakos, G. *Cross-Validation.* August 2018. Online; visited 16.11. 2019. Available at: https://towardsdatascience.com/cross-validation-70289113a072.

[12] D'Agostino, M. and Dardanoni, V. What's so special about Euclidean distance? *Social Choice and Welfare.* Springer. 2008, p. 211–233. DOI: 1-59593-518-5.

[13] Eckersley, P. *How Unique Is Your Web Browser?. In: Atallah M.J., Hopper N.J. (eds) Privacy Enhancing Technologies. PETS 2010. Lecture Notes in Computer Science, vol 6205.* Springer, Berlin, Heidelberg, 2010. ISBN 978-3-642-14526-1.

[14] Han, J., Kamber, M. and Pei, J. *Data Mining: Concepts and Techniques.* 3rd ed. Morgan Kaufmann, 2012. ISBN 978-0-12-381479-1.

[15] Harrison, O. *Machine Learning Basics with the K-Nearest Neighbors Algorithm.* September 2018. Online; visited 14.12. 2019. Available at: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[16] Hoare, J. *Machine Learning: Pruning Decision Trees.* July 2017. Online; visited 31.12. 2019. Available at: https://www.displayr.com/machine-learning-pruning-decision-trees/.

[17] Jang Jaccard, J. and Nepal, S. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences.* Elsevier Inc. 2014, p. 973–993. ISSN: 0022-0000.

[18] Khadilkar, M., Feamster, N., Sanders, M. and Clark, R. Usage-Based Dhcp Lease Time Optimization. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement.* New York, NY, USA: Association for Computing Machinery, 2007, p. 71–76. IMC '07. DOI: 10.1145/1298306.1298315. ISBN 9781595939081. Available at: https://doi.org/10.1145/1298306.1298315.

[19] Kohno, T., Broido, A. and Claffy, K. *Remote Physical Device Fingerprinting.* IEEE, 2005. ISSN 1545-5971.

[20] Kumpošt, M. *Context information and user profiling.* Masaryk University. Supervisor Václav Matyáš, Brno, 2009. Ph.D. thesis.

[21] Kumpošt, M. and Matyáš, V. *User Profiling and Re-identification: Case of University-Wide Network Analysis. In: Fischer-Hübner S., Lambrinoudakis C., Pernul G. (eds) Trust, Privacy and Security in Digital Business. TrustBus 2009. Lecture Notes in Computer Science, vol 5695.* Springer, Berlin, Heidelberg, 2009. ISBN 978-3-642-03747-4.

[22] Kurose, J. F. and Ross, K. W. *Computer Networking a Top-Down Approach.* 5th ed. Addison-Wesley, 2010. ISBN 978-0-13-607967-5.

[23] Lanze, F., Panchenko, A., Braatz, B. and Zinnen, A. *Clock skew based remote device fingerprinting demystified.* IEEE, 2012. ISBN 978-1-4673-0919-6.

[24] Laperdrix, P., Bielova, N., Baudry, B. and Avoine, G. Browser Fingerprinting: A Survey. *ACM Trans. Web.* New York, NY, USA: Association for Computing Machinery. april 2020, vol. 14, no. 2. DOI: 10.1145/3386040. ISSN 1559-1131. Available at: https://doi.org/10.1145/3386040.

[25] Lee, M. S., Rhee, J., Kim, B. and Zhang, B. AESNB: Active Example Selection with Naïve Bayes Classifier for Learning from Imbalanced Biomedical Data. In: *2009 Ninth IEEE International Conference on Bioinformatics and BioEngineering*. 2009, p. 15–21.

[26] Manning, C. D., Raghavan, P. and Schütze, H. *Introduction to Information Retrieval*. 1st ed. Cambridge University Press, 2008. ISBN 0521865719.

[27] Mayer, J. R. *"Any person... a pamphleteer" Internet Anonymity in the Age of Web 2.0*. Princeton University, Faculty of the Woodrow Wilson School of Public and International Affairs., Princeton, 2009. Bachelor's thesis.

[28] McInerney, D. O. and Nieuwenhuis, M. A comparative analysis of kNN and decision tree methods for the Irish National Forest Inventory. *International Journal of Remote Sensing*. Taylor & Francis. 2009, vol. 30, no. 19, p. 4937–4955. DOI: 10.1080/01431160903022936. Available at: https://doi.org/10.1080/01431160903022936.

[29] McKinney, W. Data Structures for Statistical Computing in Python. In: Walt, S. van der and Millman, J., ed. *Proceedings of the 9th Python in Science Conference*. 2010, p. 51 – 56.

[30] Mockapetris, P. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION* [Internet Requests for Comments]. RFC 1035. RFC Editor, 1987. Available at: https://www.rfc-editor.org/rfc/rfc1035.txt.

[31] Murdoch, S. J. Hot or Not: Revealing Hidden Services by their Clock Skew. *Computer and Communications Security*. Association for Computing Machinery. 2006, p. 27–36.

[32] Odom, W. *CCENT/CCNA ICND 1 100-105 Cert Guide*. 1st ed. CiscoPress, 2016. ISBN 978-1587205804.

[33] Phillips, G. *What Are Supercookies? Here's How to Remove Them Properly*. August 2019. Online; visited 14.12. 2019. Available at: https://www.makeuseof.com/tag/what-are-supercookies-and-why-are-they-dangerous/.

[34] Polčák, L. *Lawful Interception: Identity Detection*. Brno University of Technology, Faculty of Information Technology. Supervisor Švéda Miroslav, Brno, 2017. Ph.D. thesis.

[35] Ratan, V. and Li, K. F. *NetFlow: Network Monitoring and Intelligence Gathering*. 1st ed. Springer, 2016. ISBN 978-3-319-49108-0.

[36] Release, R. N. P. *The RIPE NCC has run out of IPv4 Addresses*. November 2019. Online; visited 9.1. 2020. Available at: https://www.ripe.net/publications/news/about-ripe-ncc-and-ripe/the-ripe-ncc-has-run-out-of-ipv4-addresses.

[37] Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3* [Internet Requests for Comments]. RFC 8446. RFC Editor, 2018. Available at: https://www.rfc-editor.org/rfc/rfc8446.txt.

[38] SANTOS, O. *Network Security with NetFlow and IPFIX*. Paul Boger, 2016. ISBN 978-1-58714-438-7.

[39] SHARMA, N. *Ways to Detect and Remove the Outliers*. May 2018. Online; visited 11.4. 2020. Available at: https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba.

[40] TOUSSIANT, M. *Introduction to Machine learning*. 2019. Online; visited 14.12. 2019. Available at: http://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/19-MachineLearning/paper.pdf.

[41] VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., HABERLAND, M., REDDY, T. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, vol. 17, p. 261–272. DOI: 10.1038/s41592-020-0772-5. Available at: https://doi.org/10.1038/s41592-019-0686-2.

[42] WEISS, N. A. *Introductory Statistics*. 10th ed. Pearson Education, 2017. ISBN 9780321989178.

[43] XU, S. Bayesian Naïve Bayes classifiers to text classification. *Journal of Information Science*. 2018, vol. 44, no. 1, p. 48–59. DOI: 10.1177/0165551516677946. Available at: https://doi.org/10.1177/0165551516677946.

[44] YE, J. Cosine similarity measures for intuitionistic fuzzy sets and their applications. *Mathematical and Computer Modelling*. ScienceDirect. 2011, p. 91–97. ISSN: 0895-7177.

[45] ZANDER, S. and MURDOCH, S. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In:. January 2008, p. 211–226.

[46] ZENDULKA, J., BARTÍK, V., LUKÁŠ, R. and RUDOLFOVÁ, I. *aj.: Získávání znalostí z databází, Studijní opora*. 2010.

[47] ZHAO, Y., NASRULLAH, Z. and LI, Z. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research*. 2019, vol. 20, no. 96, p. 1–7. Available at: http://jmlr.org/papers/v20/19-011.html.

[48] ZOU, K., WARFIELD, S., BHARATHA, A., TEMPANY, C., KAUS, M. et al. Statistical Validation of Image Segmentation Quality Based on a Spatial Overlap Index. *Academic radiology*. february 2004, vol. 11, p. 178–89. DOI: 10.1016/S1076-6332(03)00671-8.

Appendix A

Data-set Description

This chapter contains a detailed description of the data-sets used in this thesis. The data used throughout this thesis is provided by Cisco Systems and is collected using NetFlow (described in Section 2.2), DNS queries (described in Section 2.3), and lastly, HTTP and TLS fingerprinting methods (described in Sections 2.3 and 2.4 respectively).

The data-set contains real life data, therefore, the data itself cannot be published. The data-set is stored in a tab separated format (shortly .tsv), where columns are separated by the symbol "`\t`" and each new entry is on a new line. This format has been chosen over the comma separated value (shortly .csv) format, since the HTTP fingerprint can contain the character "`,`", which would create additional columns and break the pattern of the data in which the features are saved.

Since network administrators have different goals at what information they want to collect about their devices and users, some of the information such as HTTP and TLS fingerprints are not always available. Missing fingerprints and other features might also be caused by the use of different protocols, such as TACACS+ or RADIUS, which do not generate a HTTP fingerprint. Therefore, this thesis proposes a method that is robust and is able to overcome the shortcoming of unavailable features. Information about how often and what features are available in the data-sets are described in Sections A.1 and A.2. Features, such as the TLS fingerprint, are represented by hashes of their real fingerprints to reduce their size.

There are two data-sets used within this thesis, differing in the availability of one feature and the length of the export. Another difference is the number of devices. The first, shorter, data-set was used in the initial stages of the experiments to evaluate the feasibility of the newly proposed method. Afterwards, when the proposed method seemed to yield useful results, the larger and more comprehensive data-set was tested. A fictional example of the data can be found on the attached DVD, whose contents are described in the Appendix E.

A.1 Data-set 1 Description

This section contains a description of the first data-set used in this thesis. The following Tables A.1 and A.2 describe the general information and available features respectively regarding `Data-set 1`. This data-set contains a wide variety of available data and provides good starting point for network device identification methods.

Total flows	4,828,868
Length of export [h]	8
Total devices	4,555
Availability of URL [%]	3.6
Availability of TLS fingerprint [%]	3.5
Availability of HTTP fingerprint [%]	0.3

Table A.1: Basic information regarding Data-set 1.

Feature	**Total unique values**
Timestamp	28,800
MAC address	4,555
Source IP address	4,768
Destination IP address	26,459
Port	15,747
URL	23,691
TLS fingerprint	166,867
HTTP fingerprint	230

Table A.2: Basic information regarding features of Data-set 1.

A.2 Data-set 2 Description

This section contains a detailed description of `Data-set 2`. Unlike the first data-set, this one contains a larger array of devices and an additional feature – username. This feature has been added to emulate users owning multiple devices on the network, which correlates to the real world application of the proposed algorithm. The following Tables A.3 and A.4 summarize the general information and available features respectively regarding `Data-set 2`. The use of the additional feature is described in Section 6.4.

Total flows	64,806,407
Length of export [h]	16
Total users	7,385
Total devices	10,737
Availability of URL [%]	0.159
Availability of TLS fingerprint [%]	0.170
Availability of HTTP fingerprint [%]	0.0

Table A.3: Basic information regarding data-set 2.

Feature	**Total unique values**
Timestamp	57,600
Username	7,385
MAC address	10,737
Source IP address	4,768
Destination IP address	26,459
Port	15,747
URL	23,691
TLS fingerprint	166,867
HTTP fingerprint	230

Table A.4: Basic information regarding features of Data-set 2.

Appendix B

k-NN with Segmented Profiles Model Summary

The following Table B.1 summarizes the settings of the best performing algorithm proposed in this thesis.

Used Features	`feature_exp1`
Distance Metric	Cosine similarity
Time-window length [s]	300
Training time [s]	3600
Aggregation based on (training)	MAC address, username
Aggregation based on (classification)	Source IP address, username
k	3
Number of Stored Profiles	12
Sliding window	Yes
Outlier Detection	z-score outlier detection
New Device Threshold	0.9183

Table B.1: Summary of the settings of the best performing algorithm proposed in this thesis.

Appendix C

Excel@FIT Conference

An article about the proposed algorithm has been published as a part of the Excel@FIT 2020 conference. The following poster has been used to represent the paper during the participant exhibition of their works.

8. Behavior-Based Network Device Tracking

Michael Adam Polák
Faculty of Information Technology, Brno University of Technology, Czech republic
xpolak31@stud.fit.vutbr.cz

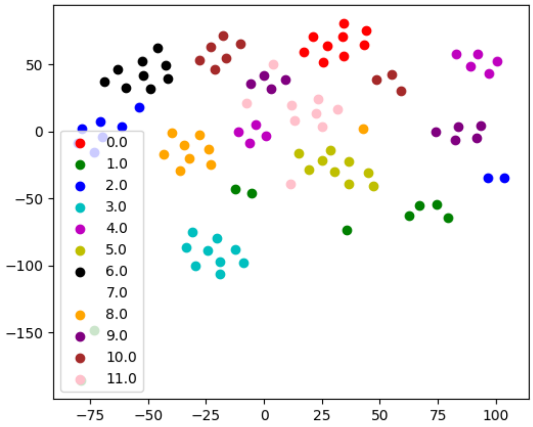
What is it?

- Identification of network devices based on their behavior
- Tracking devices with sparse availability of features
- Passive approach with low time and spatial complexity
- High accuracy compared to currently used methods

Is your device is anonymous?

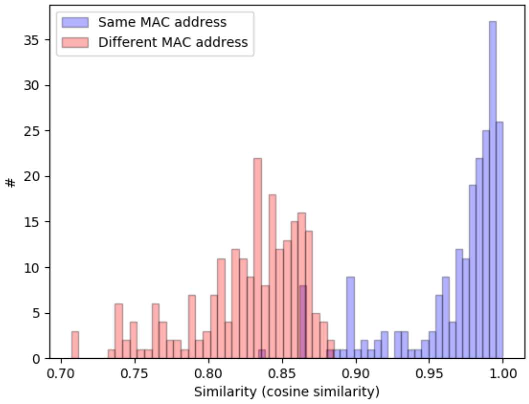
Device Profile Distribution

- Profile distribution created with t-SNE, illustrating the clustering of device profiles

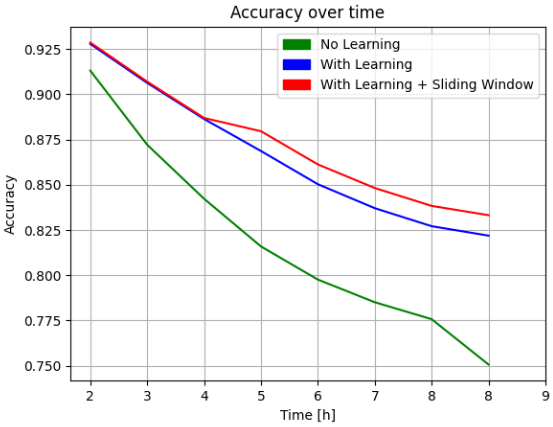


Device Profile Similarity

- Similarity scores of devices with the same MAC addresses in blue and different MAC addresses in red
- Distinct separation of similarities between devices



Achieved Accuracy over Time



Further Improvements

- Better Outlier Detection
- Parallelization

Appendix D

Application Design Diagram

The following Figure D.1 contains the class diagram describing the internal structure of the tool containing the network identification algorithms.

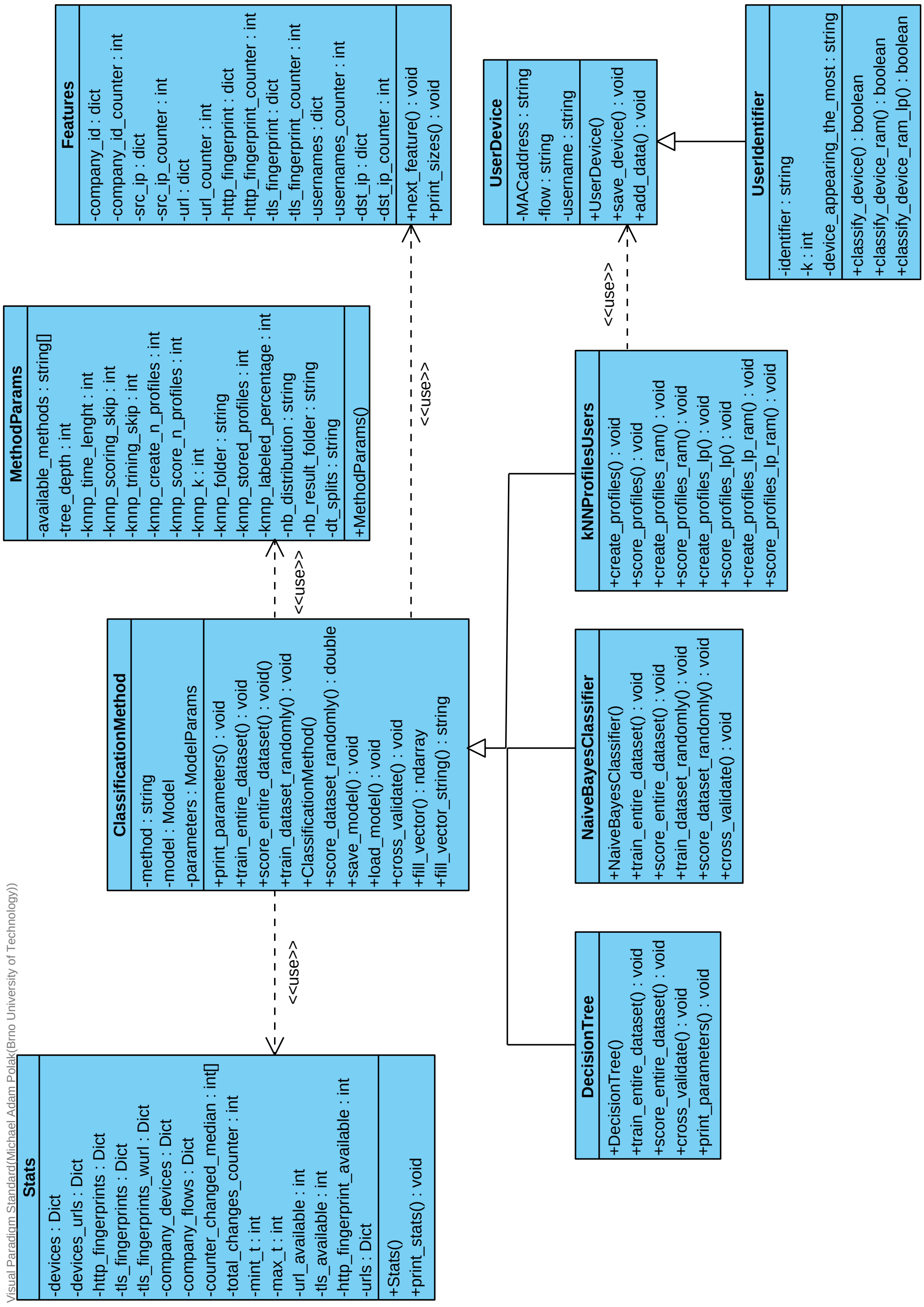

Features
-company_id : dict
-company_id_counter : int
-src_ip : dict
-src_ip_counter : int
-url : dict
-url_counter : int
-http_fingerprint : dict
-http_fingerprint_counter : int
-tls_fingerprint : dict
-tls_fingerprint_counter : int
-usernames : dict
-usernames_counter : int
-dst_ip : dict
-dst_ip_counter : int
+next_feature() : void
+print_sizes() : void
UserDevice
-MACaddress : string
-flow : string
-username : string
+UserDevice()
+save_device() : void
+add_data() : void
UserIdentifier
-identifier : string
-k : int
-device_appearing_the_most : string
+classify_device() : boolean
+classify_device_ram() : boolean
+classify_device_ram_lp() : boolean
MethodParams
-available_methods : string[]
-tree_depth : int
-knnp_time_lenght : int
-knnp_scoring_skip : int
-knnp_trining_skip : int
-knnp_create_n_profiles : int
-knnp_score_n_profiles : int
-knnp_k : int
-knnp_folder : string
-knnp_stored_profiles : int
-knnp_labeled_percentage : int
-nb_distribution : string
-nb_result_folder : string
-dt_splits : string
+MethodParams()
<<use>>
<<use>>
<<use>>
kNNProfilesUsers
+create_profiles() : void
+score_profiles() : void
+create_profiles_ram() : void
+score_profiles_ram() : void
+create_profiles_lp() : void
+score_profiles_lp() : void
+create_profiles_lp_ram() : void
+score_profiles_lp_ram() : void
ClassificationMethod
-method : string
-model : Model
-parameters : ModelParams
+print_parameters() : void
+train_entire_dataset() : void
+score_entire_dataset() : void()
+train_dataset_randomly() : void
+ClassificationMethod()
+score_dataset_randomly() : double
+save_model() : void
+load_model() : void
+cross_validate() : void
+fill_vector() : ndarray
+fill_vector_string() : string
NaiveBayesClassifier
+NaiveBayesClassifier()
+train_entire_dataset() : void
+score_entire_dataset() : void
+train_dataset_randomly() : void
+score_dataset_randomly() : void
+cross_validate() : void
<<use>>
DecisionTree
+DecisionTree()
+train_entire_dataset() : void
+score_entire_dataset() : void
+cross_validate() : void
+print_parameters() : void
Visual Paradigm Standard(Michael Adam Polak(Brno University of Technology))
Stats
-devices : Dict
-devices_urls : Dict
-http_fingerprints : Dict
-tls_fingerprints : Dict
-tls_fingerprints_wurl : Dict
-company_devices : Dict
-company_flows : Dict
-counter_changed_median : int[]
-total_changes_counter : int
-mint_t : int
-max_t : int
-url_available : int
-tls_available : int
-http_fingerprint_available : int
-urls : Dict
+Stats()
+print_stats() : void

Figure D.1: Class diagram describing the internal structure of the tool.

Appendix E

Contents of the Attached DVD

The following directory tree describes the hierarchy and the contents of the attached DVD to this thesis.

```
/ .................................................... Root directory of the DVD
├── source ................................ Source code of the implemented tool
│   └── install.sh ................................ Installs all necessary libraries
├── text ..................................... Text and source code of the thesis
│   ├── thesis.pdf ........................................... Text of the work
│   └── latex ........................ Source code of this thesis written in LaTeX
├── config_examples ........... Configuration file examples for each of the methods
├── dataset_example ......... Fictional data-set representing the format of the data
├── experiments ...... YAML files and anonymized confusion matrices of experiments
├── excel ......................... Materials submitted to the Excel@FIT conference
└── README.txt .................... Description of the contents of the attached DVD
```