



Assignment of bachelor's thesis

Title:	Crypto-currency miner detection from extended IP flow data
Student:	Richard Plný
Supervisor:	Ing. Karel Hynek
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2022/2023

Instructions

Study network monitoring and analysis technology based on Deep Packet Inspection and (extended) IP flows. Analyze the area of cryptocurrencies, their mining approaches, and their behavior on the computer networks focusing on the miner detection possibilities. Design an algorithm for automatic detection of crypto-miners in the network based on observed network traffic. Develop a software prototype capable of processing real high-speed network traffic using the NEMEA system [1,2]. Test and evaluate the prototype with the created dataset and data provided by the supervisor of this thesis.

[1] T. Cejka, V. Bartoš, M. Svepes, Z. Rosa, and H. Kubatova, "NEMEA: A Framework for Network Traffic Analysis," in 12th International Conference on Network and Service Management (CNSM 2016), Montreal, Canada, 2016.

[2] <https://github.com/CESNET/NEMEA>

Bachelor's thesis

**CRYPTO-CURRENCY
MINER DETECTION
FROM EXTENDED IP
FLOW DATA**

Richard Plný

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Karel Hynek
May 2, 2022

Czech Technical University in Prague
Faculty of Information Technology

© 2022 Richard Plný. Citation of this thesis.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Plný Richard. *Crypto-currency miner detection from extended IP flow data.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Contents

Acknowledgments	ix
Declaration	x
Abstract	xi
List of abbreviations	xii
Introduction	1
1 Theoretical Part	3
1.1 Cryptocurrency	3
1.1.1 Mining and mining pools	4
1.1.2 Mining protocols	5
1.1.3 Abusive mining	7
1.1.4 Previous attempts of mining detection	8
1.2 Network monitoring	9
1.2.1 Deep Packet Inspection	10
1.2.2 IDS/IPS Systems	11
1.2.3 Flow-based network monitoring	12
1.2.4 NEMEA	13
1.3 Machine Learning	14
1.3.1 Selected supervised ML algorithms	15
1.4 Dempster-Shafer theory	15
2 Datasets	17
2.1 Current situation of cryptocurrencies	17
2.2 Local examination of miners' traffic	18
2.3 Traffic capture on CESNET	18
2.4 Creating datasets	20
2.5 Summary	21
3 Analysis and design	23
3.1 Analysis of traffic from CESNET	23
3.2 Design	25
3.3 ML classifier	27
3.4 Stratum detector	28
3.5 TLS SNI Classifier	30
3.6 Meta classifier	31
3.7 Implementation	33
3.7.1 Miner aggregator	34
3.7.2 IDEA reporter	35

4 Evaluation	37
4.1 Datasets' quality	37
4.1.1 Permutation tests	37
4.2 Performance metrics	38
4.3 Performance of the miner detector	39
4.4 Deployment on CESNET	41
Conclusion	43
A Selected mining pools	45
B Permutation test results	47
C ML models' results	51
D Experimental credibility values	55
E User manual	57
Contents of enclosed CD	65

List of Figures

1.1	Blockchain P2P network	4
1.2	Blockchain structure	4
1.3	Comparison of solo and pooled mining	5
1.4	Stratum requests sent between a miner and a mining pool	6
1.5	Event-based DPI	11
1.6	Flow-based network monitoring architecture	13
1.7	Network monitoring setup with NEMEA	14
1.8	Internal architecture of NEMEA	14
1.9	Selected ML models	16
2.1	Scheme of the Python script for rule generation	19
3.1	Unencrypted flows in CESNET traffic	23
3.2	Statistics of exchanged bytes	24
3.3	Statistics of exchanged packets	25
3.4	Packet characteristics	25
3.5	Ratios of received and sent packets in flows	26
3.6	Ratios of TCP PUSH flag in flows	26
3.7	Overall flows duration	27
3.8	High-level concept of the Meta classifier	27
3.9	Initial design of the Stratum detector	29
3.10	Design of TLS SNI classifier	31
3.11	Schema of the Meta classifier	32
3.12	Proposed deployment architecture of our miner detector	34
3.13	Screenshot of IDEA alert with a potential miner in the Mentat web interface	35
4.1	ROC AUC metrics	39
4.2	Confusion matrices of miner detector (the Meta classifier)	40
4.3	Confusion matrices of support classifiers when all data were used	40
4.4	Confusion matrices of support classifiers when the pre-filter was in place	41
B.1	Performance drops of ML models used for permutation testing	49

List of Tables

2.1	Datasets overview	20
3.1	Overview of flows' metrics exported on CESNET network	24
3.2	Overview of selected features for ML models	28
3.3	Hyperparameters of the best Random forest model	28
4.1	Performance of the chosen Random forest model	39
A.1	Selected mining pools for BTC	45
A.2	Selected mining pools for ETH	46
A.3	Selected mining pools for XMR	46
B.1	<i>P</i> -values for dataset 01 from permutation tests with 200 permutations	47
B.2	<i>P</i> -values for dataset 02 from permutation tests with 200 permutations	48
B.3	<i>P</i> -values for dataset 03 from permutation tests with 200 permutations	48
B.4	<i>P</i> -values for dataset 04 from permutation tests with 200 permutations	48
C.1	Overview of Decision tree models	51
C.2	Overview of Decision tree models' performance	52
C.3	Overview of Random forest models	52
C.4	Overview of Random forest models' performance	53
D.1	Experimental credibility values for Stratum classifier	55
D.2	Experimental credibility values for TLS SNI classifier	55

List of code listings

1	Stratum request	7
2	Stratum request used in active probing	19
3	Bash command used for conversion of a trapcap file into csv	20
4	Complete Regex pattern for matching Stratum request and notification	30
5	Complete Regex pattern for matching Stratum response	30
6	Concatenation of keywords to create one Regex pattern	31
7	Conjunctive combination of mass functions to get pignistic function	33

I am deeply grateful to my supervisor Ing. Karel Hynek for his professional guidance and help, constant belief in me and all the devoted hours. I would also like to thank Ing. Tomáš Čejka, Ph.D. and Ing. Dominik Soukup for their guidance and advice on many new technological areas which was very valuable when writing this thesis. In addition, I appreciate the friendly attitude of members of FIT CTU, FIT BUT and CESNET with whom I had the opportunity to collaborate with. Last but not least, I would like to thank my family and friends for their support and patience during my studies, which I am sure have not always been easy.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the “Work”), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Praze on May 2, 2022

.....

Abstrakt

Tato bakalářská práce se zaměřuje na těžbu kryptoměn z bezpečnostní perspektivy s důrazem na nelegální těžbu. Zkoumá možnosti detekce těžení kryptoměn ve vysokorychlostních počítačových sítích na úrovni monitorování síťových toků. Práce obsahuje návrh platformy pro kontinuální záchyt komunikace, která je použita k vytvoření datových sad obsahující komunikaci těžících softwarů z reálného provozu. Dále je navržena metoda detekce, která je schopna provozu i na vysokorychlostních sítích. Navržené řešení je implementováno jako skupina modulů systému NEMEA. Tato skupina modulů byla nasazena a testována na národní síti provozované sdružením CESNET.

Klíčová slova kryptoměna, miner, mining pool, detekce, monitorování sítě

Abstract

This bachelor thesis addresses cryptomining from the security perspective with an emphasis on abusive mining. It explores the possibilities of detection of cryptominers in high-speed computer networks using a flow-based monitoring approach. A setup for continuous traffic capture is proposed and used for creating datasets with real-world miners' traffic. Furthermore, a detection method is proposed, capable of operation on high-speed networks. The proposed solution was implemented as a group of NEMEA modules. Moreover, it was deployed and evaluated on the national network CESNET2 operated by CESNET.

Keywords cryptocurrency, miner, mining pool, detection, network monitoring

List of abbreviations

AB	AdaBoost
AEAD	Authenticated Encryption with Associated Data
API	Application Programming Interface
AUC	Area under the ROC curve
BTC	Bitcoin
DASH	Dash
DNS	Domain Name System
DPI	Deep Packet Inspection
DST	Dempster-Shafer Theory
DoH	DNS over HTTPS
DoT	DNS over TLS
ETH	Ethereum
FN	False Negative
FP	False Positive
FQDN	Fully Qualified Domain Nam
IDS	Intrusion Detection System
IPFIX	Internet Protocol Flow Information Export
IPS	Intrusion Prevention System
JS	JavaScript
L7	Layer 7 of ISO/OSI model
ML	Machine Learning
MLP	Multi-layer Perceptron
NEMEA	Network Measurements Analysis
NIC	Network Interface Controller
P2P	Peer-to-peer
PCA	Principal Component Analysis
Regex	Regular Expression
ROC	Receiver Operating Characteristic
RVN	Ravencoin
SNI	Server Name Indication
SSL	Secure Sockets Layer
SVM	Support Vector Machine
TAP	Test Access Port
TBM	Transferable Belief Model
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
UNSC	United Nations Security Council
URL	Uniform Resource Locator
XMR	Monero
ZEC	Zcash

Introduction

Cryptocurrencies have become an essential part of the financial market and part of daily life for many of us. Some of us invest and trade cryptocurrencies, and some of us also *mine* them on our personal computers, meaning that we are using electrical power in exchange for a reward. Some people might even use company computers, computers belonging to someone else, or cloud resources which sometimes goes against the policy [1]. Computer resources may also be stolen by malware and used for cryptomining at the victims' expenses without them even knowing about it. Attackers may also use malvertising [2] to hijack victims' browsers to mine Monero and other cryptocurrencies [3, 4, 5]. Detection of cryptomining can be used as a defense mechanism and is needed with more and more attacks.

This thesis was developed in coordination with CESNET¹ which expressed the need to protect its computational resources against abusive mining. CESNET operates several 100 Gbps networks, but unfortunately cryptomining detection has not been studied very thoroughly on networks with such speed so far. Additionally, this thesis is meant for network operators who want to protect their networks against misusing property and wasting electrical power. Detection of cryptomining in a network may also indicate that the network was attacked and compromised.

We are building our research on the previous work of V. Veselý and M. Žádník and lots of others who researched network monitoring, traffic inspection, and detection of miners' communication. The main goal of this thesis is to design and implement a reliable detector of cryptomining with a low false-positive rate. Also, deploy this detector on the CESNET network and evaluate its outputs. Another goal is to examine the current cryptocurrency situation and create datasets with example communication for further research.

We focused on how cryptocurrencies and cryptomining work in general and from the point of view of network monitoring in chapter 1. Furthermore, network monitoring approaches and previous attempts to detect cryptomining are discussed. Chapter 2 examines the current cryptocurrency situation and mining pools. Miner traffic from a virtual machine is captured and analyzed. Moreover, it describes how traffic was captured on the CESNET network. We thoroughly analyzed captured traffic, proposed the design and described the implementation of our detector in chapter 3. Evaluation of detector's results is in chapter 4.

¹www.cesnet.cz

Theoretical Part

This chapter provides a theoretical background for later work. Basics of cryptocurrencies, mining process and malicious use of cryptocurrencies are discussed. In addition to this, we examined previous attempts at detection of cryptomining. Network protocols necessary for mining, introduction to network monitoring and its tools are described. Moreover, ground zero ideas of machine learning are summed up with an overview of selected machine learning models. Lastly, the essentials of a mathematical theory for a combination of probabilities are presented, called the Dempster-Shafer Theory.

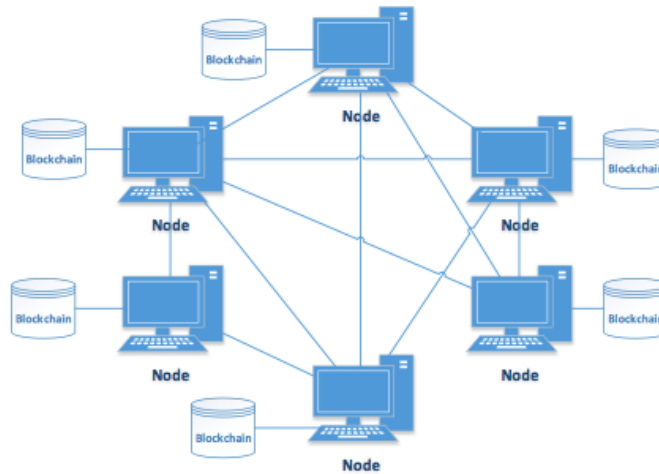
1.1 Cryptocurrency

Cryptocurrencies are digital money based on decentralization, strong cryptography, and Blockchain technology. The first and most famous cryptocurrency, Bitcoin, originated in 2008 when Satoshi Nakamoto published his whitepaper called “Bitcoin: A Peer-to-Peer Electronic Cash System” [6]. From that point, many new cryptocurrencies appeared. According to Phipps [7], 7% of the world’s money, \$2.48 trillion, is involved in cryptocurrency to the 24.01.2022.

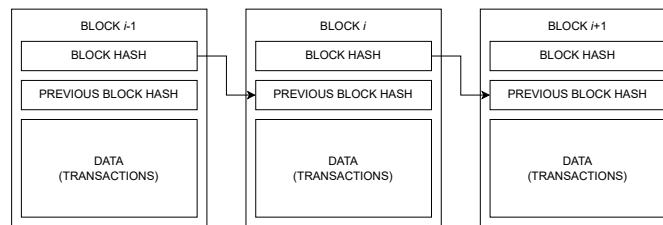
Based upon Nakamoto [6], cryptocurrencies use distributed peer-to-peer networks, shown on the figure 1.1 in which there is no central point or machine, nodes of a network communicate directly with each other. Thus, two willing parties can transact directly with each other without the need for a trusted third party [6]. Cryptography is used to achieve trust in such environment.

Crosby et al. [9] define Blockchain as “a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties”. Each block has a timestamp, a group of transactions, and a parent block hash which creates the reference and forms a chain (shown on the figure 1.2). The first block of this chain is called *genesis block* and has an empty reference to the parent block. Moreover, entered records cannot be erased [9].

Even though cryptocurrencies use wallet addresses, it is still possible to follow transactions in Blockchain and piece together someone’s information [10]. Privacy coins, discussed by Hayward [10], are types of cryptocurrencies that use several cryptographic techniques to protect user details. Some examples are Monero (XMR), Zcash (ZEC) and Dash (DASH). Thanks to this, users can choose which information will be shared with external parties. On the contrary, private coins became widely used in ransomware attacks and other illegal activities and for this reason, privacy coins have been banned in Japan and South Korea [10, 11].



■ **Figure 1.1** Blockchain P2P network, taken from [8]



■ **Figure 1.2** Blockchain structure

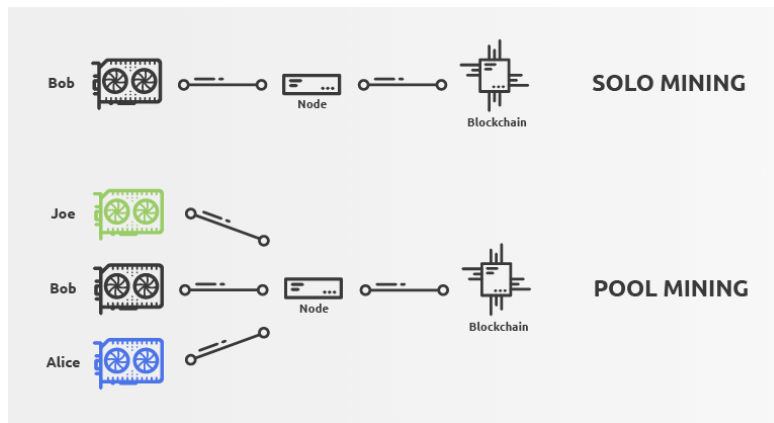
1.1.1 Mining and mining pools

Ghimire et al. [12] described miners as “individuals who secure cryptocurrency’s network”. Mining is described as “the process of adding transaction records to cryptocurrency’s public ledger of past transactions or Blockchain” [12]. This process involves solving a puzzle — a hard mathematical problem. Mining of a new block is rewarded by obtaining coins of the cryptocurrency (either new coins or transaction fees). Thus, a miner uses electrical power in exchange for a reward. Two types of mining were described by Tarman [13] – *solo* and *pooled*.

Tarman [13] described solo mining as “an attempt to confirm blocks of transactions on the Blockchain alone, as an individual miner”. Miner will get block rewards and transaction fees all by himself — large payments within longer intervals. Miners who are mining solo have to communicate directly with the cryptocurrency’s network. Unfortunately, there is a high chance that solo mining will not produce any reward [13].

The other type of mining described in [13] is pooled mining. Miners connect to a mining pool and share resources in order to mine blocks more often, see figure 1.3. Rewards for mined blocks are split, providing smaller but steady payments. Usually, splits are based on the work done by a miner, but exact settings depend on the pool operator [14]. This allows the miner to get a reward even if he is not the one who generated the new block. Pool’s strength is measured in the overall hash rate — “combined computational power that is being used to mine and process transactions” [15]. It can also be interpreted as the number of hashes a pool is able to generate per second.

Based on Bitcoin Developer Guides [14], the mathematical problem that miners must solve in



■ **Figure 1.3** Comparison of solo and pooled mining, taken from [13]

the case of Bitcoin is to find a hash of a nonce (number used only once) and a block header. For this hash to be valid and accepted, it has to be below the target threshold, meaning that this hash has to start with a certain number of leading zeroes (based on the difficulty). We will now focus on the process of solo mining. Firstly, new transactions are fetched from the network, and a block header is generated. Miner then starts calculating hashes from the block header and every possible value of the nonce. If a valid hash is found, the block is completed (or mined) and broadcasted to the network for others to add it to their Blockchain.

Pooled mining described in [14] is very similar but has one main difference. The miner does not get new transactions from the network but connects to a mining pool and asks for work. The mining pool will reply with data necessary for mining. When the miner finds a valid hash, he will notify the pool. Difficulty in a mining pool is usually set slightly below the network’s difficulty. This causes miners to send results to a pool more often. The majority of these hashes are not valid but are used to prove that miner did his share of work. Such messages are called *share messages* [14]. Some hashes will also be below the network’s difficulty — a new block is found and broadcasted to the network by the mining pool. The mechanism of share messages can also be used to split rewards and fees to miners based on what percentage of shares they did.

1.1.2 Mining protocols

Plenty of mining software is available [16]. Each mining software implements a specific algorithm for a cryptocurrency it is meant to mine. Moreover, it has to implement network protocols for communication with a mining pool such as Getwork or Stratum [14, 17, 18]. Other network protocols are needed for communication or directly used by mining protocols, such as TCP, DNS and TLS [14, 19]. P2P Network is used by solo miners and mining pools for communication with the whole cryptocurrency network [14].

Transmission Control Protocol (TCP), invented by Cerf et al. [20], is “a connection-oriented, end-to-end reliable protocol” [21]. It is the most popular protocol for reliable and connection-oriented data transfer from the transport layer [22]. This protocol is used for ordered and error-checked delivery of a stream of bytes. Connection is established by a three-way handshake before sending any application data.

Transport Layer Security (TLS) is a protocol from the application layer whose primary goal is to “provide privacy and data integrity between two communicating applications” [23]. It is also a successor of the Secure Sockets Layer (SSL) [24]. As described in [23], TLS connection is private — data are encrypted via symmetric cryptography and unique keys are used for each connection.

```
#1 { "worker": "eth1.0", "jsonrpc": "2.0", "params":
  ["0x9c99d212f7e5daa18ab50810e0fd255df04303b/tester.worker1/vvesely@mailinator.com",
  "x"], "id": 2, "method": "eth_submitLogin"}
#2 {"jsonrpc": "2.0", "id": 2, "result": true}
#3 {"jsonrpc": "2.0", "id": 0, "result": ["0x415559c31768833f25b6dbfbb39be72e71375e14a3a711e
589696850bc9431ec", "0x71a56feffb6f10ea9d76e1a9464eb0abd86e4349ae98fb794923a65b650282
a3", "0x00000000dbe6fecebdedd5beb573440e5a884dlb2fbf06fce912adcb8d8422e"]}
#4 {"worker": "", "jsonrpc": "2.0", "params": [], "id": 3, "method": "eth_getWork"}
#5 {"jsonrpc": "2.0", "id": 0, "result": ["0x41f3161ce643ebcf25d34dde1ac0d3e695edcbf136eed96
680bac4cf1a82e417", "0x71a56feffb6f10ea9d76e1a9464eb0abd86e4349ae98fb794923a65b650282
a3", "0x00000000dbe6fecebdedd5beb573440e5a884dlb2fbf06fce912adcb8d8422e"]}
#6 {"id": 4, "method": "eth_submitWork", "params": ["0x07a05fa4133e5126", "0x41f3161ce643ebcf
25d34dde1ac0d3e695edcbf136eed96680bac4cf1a82e417", "0x44e9206e9c830706f60e0129bdd117a
2718d6553f3405b477541994df3583d4b"]}
#7 {"jsonrpc": "2.0", "id": 4, "result": true}
#8 {"jsonrpc": "2.0", "id": 0, "result": ["0x379dd042dcd4954143b4f2d4a35b8db62f503be23f012a
4b1bd6a52dbd44c78", "0x71a56feffb6f10ea9d76e1a9464eb0abd86e4349ae98fb794923a65b650282
a3", "0x00000000dbe6fecebdedd5beb573440e5a884dlb2fbf06fce912adcb8d8422e"]}
#9 {"id": 6, "jsonrpc": "2.0", "method": "eth_submitHashrate", "params": ["0x1d07cc6",
  "0x0000000000000000000000000000000000000000000000000000000000002df91be"]}
  {"worker": "", "jsonrpc": "2.0", "params": [], "id": 3, "method": "eth_getWork"}
```

■ **Figure 1.4** Stratum requests sent by a miner to a mining pool (blue) and vice versa (red) [27]

Moreover, the TLS connection is reliable since it is built on the TCP.

Domain Name System (DNS) is meant to “provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations” [25]. It is used for resolving FQDNs (fully qualified domain names) of mining pools to their IP addresses. Moreover, DNS over TLS (DoT) or DNS over HTTPS (DoH) can be used [26]. DNS requests are wrapped and encrypted by TLS or HTTPS to make them private and secure.

As described in Bitcoin Developer Guides [14], Getwork RPC was the first communication protocol used by mining pools and miners. This protocol is constructing block headers for miners directly. Modern miners need to make hundreds of requests per second. We will not discuss this protocol since it is deprecated [14]. GetBlockTemplate RPC was an improved mining method [14]. As described by Žádník et al. [27], block creation is done by miners instead of pools, creating a more decentralized environment. It also reduces mining protocol overhead.

Stratum (Stratum V1), described in [17], is currently the most frequently used mining protocol. It was introduced in 2012 and was firstly implemented on Bitcoin.cz Mining Pool (called Slush Pool nowadays) [17]. Stratum uses plain TCP sockets where packet payloads are JSON messages (based on JSON RPC 2.0¹) with “\n” at the end. There are three message types — request, response, and notification. The structure of typical JSON carried by Stratum is shown on the listing 1. Moreover, communication between a miner and a mining pool is shown on the figure 1.4, blue requests are sent by a miner to a mining pool and red requests are vice versa (a mining pool to a miner).

Stratum network protocol specification [28] defines two formats of messages — request and response. Every RPC request has the following fields:

- ID — integer, string or null
- method — unicode string
- parameters — list of parameters

Moreover, requests can be of two types. The first one is part of the RPC standard, this type expects a response. The other type of request is called *notification* and does not expect a response. These two types can be distinguished by the value of *id*, notification has *id* set to null.

¹www.jsonrpc.org/specification

```
{
  "jsonrpc": "2.0",
  "method": "job",
  "params":
  {
    "blob": "...",
    "job_id": "687",
    "target": "f3220000",
    "height": 2470181,
    "seed_hash": "...",
    "next_seed_hash": ""
  }
}
```

■ **Code listing 1** Stratum request (values in *blob* and *seed_hash* are omitted)

Stratum response has the following fields:

- ID — same ID as in request (for pairing request-response)
- result — any JSON encoded result
- error — null or list (error code, error message)

Stratum V2, defined in [18], is the Stratum’s direct successor. JSON RPC 2.0 was replaced by a binary format to reduce overhead. Therefore, messages are no longer human-readable. The size of a typical share message in V1 is approximately 100 bytes compared to 32 (48 if encrypted) bytes in V2. Moreover, the new version implements AEAD (authenticated encryption with associated data) to prevent Man-in-the-Middle (MITM) attacks. It is currently implemented in Braiins OS and Braiins OS+, and it is expected that Stratum V2 will be a new open standard in mining [18].

The last mentioned protocol above, the P2P Network protocol, is used to “collaboratively maintain a peer-to-peer network for block and transaction exchange” [14]. This protocol is used for distributing Blockchain and transactions. However, networking rules are not covered and therefore alternative protocols may be used. Since we are mainly focusing on pooled mining, we will not discuss these protocols in detail.

1.1.3 Abusive mining

Cryptomining is a highly competitive process since you need to be the first to receive a reward and therefore more hashes you are able to calculate, you have a higher chance of being the first one. Mining malware or illicit cryptomining refers to mining carried out by criminals using resources stolen from their victims [29]. It is a way to use more devices, increase your overall hash rate and the chance for a reward. Especially Monero became very popular when it comes to mining malware since it is known to be hard to trace [10]. Nearly 5% of Monero coins (with a value of almost \$40 million at that time) in circulation in 2018 were mined by malware [30]. Specialists from McAfee [31] reported that “coin miner malware” grew more than **4000%** in the year 2018.

Pastrana and Suarez-Tangil spent 12 years on their work [29] analyzing mining malware. They described two possible types of mining malware:

- Browser-based cryptomining
- Binary-based cryptomining

Browser-based cryptomining, also called *cryptojacking*, uses scripts to run in web pages (typically JavaScript). Mining process starts when a user visits a web page. Binary-based cryptomining uses malware to infect a machine connected to the Internet and then runs a binary that handles the mining process. Attackers can gain the hash rate of a medium-sized mining farm by using hundreds of infected machines. Moreover, they say: “Overall, we estimate there are at least 2218 active campaigns that have accumulated about 720K XMR (57M USD). Interestingly just a single campaign (C#623) has mined more than 163K XMR (18M USD), accounting for about 23% of the total estimated. This campaign is still active at the time of writing.” [29].

Mining malware also targets corporate networks. A cybersecurity company called Sophos, which had over 500 000 businesses as customers last year [32] published a detailed report [33] of how mining malware gets into networks. We also found an example of cryptojacking on a larger scale. UN Security Council (UNSC) found a malware mining Monero that sent mined coins to the servers located at Kim Il Sung University in Pyongyang. The Republic of Korea Financial Security Institute attributed a similar cryptojacking attack on a South Korean company [34].

1.1.4 Previous attempts of mining detection

The study performed by Jingqiang et al. [35] focused on the detection of browser-based “silent miners”. Their method uses a sandbox for loading a page and then analyzes the website’s resources to detect Javascript (JS) miners. Liu et al. [36] even uses electricity consumption data to recognize Bitcoin miners. Swedan et al. [37] proposed Mining Detection and Prevention System (MDPS) based on the mitmproxy. Their proposed solution uses URL blacklists, detection of mining code and VirusTotal API for further URL investigation. Kharraz et al. [38] inspected cryptojacking libraries in their work. JS compilation time, JS engine execution time, garbage collection and other statistics were used as features for ML models. The best model (SVM) had above 95% true positive rate. However, the papers mentioned in this paragraph are not relevant for the purposes of this thesis since they use different approaches and are not further discussed.

Muñoz et al. [39] presented a machine learning method that is able to detect cryptocurrency miners using NetFlow/IPFIX network measurements. The presented method does not need to inspect packets’ payload but still achieves similar accuracy as the DPI-based techniques. Captured traffic was analyzed and it was determined that miner flows are long duration and have a small number of transferred packets. Moreover, a server typically sends 20 times more data than a client. Models were then trained on several features:

1. Inbound and outbound packets/seconds
2. Inbound and outbound bits/seconds
3. Inbound and outbound bits/packet
4. Bits_inbound/bits_outbound ratio
5. Packets_inbound/packets_outbound ratio

The best model was Naive Bayes which achieved an average accuracy of 96.3%.

Žádník et al. [27] examined two approaches for miners’ communication detection in their paper. The first discussed approach combines active and passive detection to learn a list of existing

mining pools slowly. The second approach contains a web application used as a catalog of existing mining pools publicly available to anyone for querying.

The first discussed approach by Žádník et al. [27] combines passive detection and a secondary verification of false positives by active probing. A feature vector is created from flow data and an ML-based detector decides if flow looks like the miner’s communication. Many false positives can occur at this stage due to the heuristic nature. Active probing is then used to verify if a server where the client is connecting is really part of a mining pool structure. Several packet traces of miners connecting to well-known mining pools were analyzed in order to select features for ML. During this analysis, it was discovered that miners’ traffic has the following characteristics:

1. Mutual communication between a miner and a mining server often lasts for several hours
2. Packets are generally small, often in the range from 40 to 120 bytes
3. Most flows are observed with TCP ACK and PUSH flags set
4. The destination port is either a well-known port of a different service or not well-known but definitely lower than the source port
5. Flows are generally long-lasting, often exported before its end due to an active timeout
6. Communication is not disrupted, i.e., most flows do not contain the RST flag

However, it was determined that this design has performance issues, primarily because of the active probing.

The second discussed approach by Žádník et al. [27] describes a web application containing meta-information about mining pools. This catalog is available to the public and anyone can query the database – the name of the pool and its URL, the list of pool servers (FQDN and ports), list of IPv4 and IPv6 addresses (gained by resolving FQDNs). Basic mining pool information is collected manually by the application’s operators. The list of resolved IP addresses is automatically renewed every day. Moreover, the application supplies data for the detector from the first approach, and results from active probing are stored in the application’s database. The systems described in both steps work together to keep an up-to-date list of mining pools.

1.2 Network monitoring

As proposed in [40], we may monitor computer networks from different points of view. Network monitoring focuses on the status and performance of a network. It detects malfunctioning devices, overloaded resources, etc. Three main metrics are measured — availability, performance and configuration [40, 41]. As an alternative, network security monitoring is used to secure a network, protect transmitted data and prevent downtime. Moreover, it is “a detection-oriented and response-based approach to protecting against intrusions and vulnerabilities” [41]. Network security monitoring must also detect intrusions, attacks, anomalies and send alerts for the manual investigation to network administrators. Both types of monitoring use several approaches described later in this chapter.

Sihyung Lee et al. [42] analyzed existing network monitoring principles, their issues and possible future directions. They state that monitoring is crucial for managing networks and can be used for many critical tasks. The major function of network monitoring is the early identification of trends and patterns in both network traffic and devices. Monitoring can help network operators to find vulnerabilities in servers, limit the specific type of traffic to avoid dropping packets of other types and more. According to their work, late detection can lead to prolonged service disruption and financial losses up to millions of dollars. Moreover, they described two types of network monitoring approaches — *active* or *passive*.

Active approaches described in [42] inject the test traffic into a network to perform a measurement and usually run on an end-system. Thus, the impact on regular traffic needs to be minimal. The size and frequency of the active probing are used for measuring the impact. Ping and traceroute (on Windows, traceroute on Linux) are tools implementing active network monitoring approaches. This can be used to directly measure or inspect a specific incident of interest. Based on this whitepaper [43], active monitoring can also be used to simulate user behavior.

On the contrary, passive approaches, described in [42], do not inject any traffic into a network and only “eavesdrop” the traffic which already exists. This type of monitoring either runs on a dedicated device or is performed directly by network devices (routers, switches, ...). Passive approaches are non-intrusive and affect the monitored network less than the active approaches. They observe the actual behavior of a network. However, it can take a long time to observe a specific incident of interest. As pointed out here [43], data can be collected only from owned devices therefore leaving potential gaps for the complete monitoring.

Monitoring approaches are usually combined together since they both offer different measurement capabilities needed in different scenarios. Network operators can therefore use any combination which suits their needs [43].

According to Svoboda et al. [44], simple manual monitoring may be easy to use when the amount of monitored data is small. Otherwise, there may be so much information that it gets lost in the sea and it may become more technologically demanding to handle and store the data. Thus, they analyzed several network monitoring approaches, their trade-offs and more.

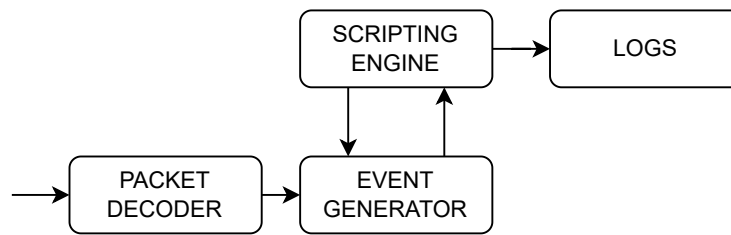
The first group of analyzed approaches by Svoboda et al. [44] is called *traffic duplication*. Traffic going through a cable is duplicated and the copy is analyzed. There are two methods of duplication — *inline* and *mirroring*. Inline duplication is done via a special device placed directly on the cable. Mirroring is done via a built-in feature of a router or a switch. Traffic mirroring can be done in several ways: port mirroring, TAP or TAP-like setup using bypass NICs.

The second group described by Svoboda et al. [44] is called *packet capture*. It allows us to save packets passing through a network interface to a file or pass them directly to network traffic analyzers. Such obtained packets are the exactly same as on the line they were captured from. Packet capture can also be viewed as a network monitoring approach consisting of the two basic steps — creating the packet capture file and performing the traffic analysis afterward. Analysis can be both automatic or manual (usually in the case of a few selected packets). Captured traffic is also usually seen as a series of IP packets (level 3 of the OSI model).

1.2.1 Deep Packet Inspection

Deep Packet Inspection (DPI) was described by Svoboda et al. [44] as an automated approach for packet inspection. Brook [45] compared DPI and conventional packet filtering, which only works with packet headers. This basic approach was dependent on the processing power of firewalls which was very low at the time. DPI not only reads headers but is also able to work with the payloads of IP packets. Network administrators and other users can set up rules for filtering spam, viruses and other malicious content. There are two types of DPI-based analysis — *pattern matching* and *event-based analysis* [44].

Furthermore, Brook [45] proposed that DPI can be used as an intrusion detection system (IDS), an intrusion prevention system (IPS) or their combination. It is a fundamental defense element capable of the prevention of spreading worms, spyware, viruses and they can even detect the usage of prohibited applications in corporate networks. It can also prevent leaking classified files and information leakage in general. Another use case is to separate traffic by priority. We can label traffic by importance and DPI can let high-priority traffic pass through before the regular one. We can also control peer-to-peer downloading — decreasing the speed of data transfer.



■ **Figure 1.5** Event-based DPI

According to Svoboda et al. [44], DPI-based analysis based on pattern matching is a method of searching byte sequences or Regex patterns through full data. It is a simple and often straightforward method but may not be enough. We may want to decode the data before the pattern matching or process them in some way, the typical example is compression. We are not able to create a Regex pattern to handle the decompression. Fortunately, monitoring devices implementing DPI can decode most protocols nowadays. Suricata² and Snort³ are implementations of pattern matching based DPI [44].

The other DPI approach described by Svoboda et al. [44] consists of the event-based analysis. Packets are processed into events that are processed by scripts afterward (shown in figure 1.5). Such scripts may implement complex algorithms and provide advanced functionality, solving shortcomings of the simple pattern matching-based DPI. Simple pattern matching is therefore replaced by more advanced programs (even though pattern matching can be implemented as such program). Moreover, algorithms can be stateful, meaning that they can remember the state between event occurrences via variables.

For the correct functionality, traffic inspected by DPI must not be encrypted. Therefore, we can only inspect traffic from unsecured sources. A possible way for inspecting encrypted traffic is using custom certificates on devices under our control, so a network node performing DPI can decrypt the traffic beforehand.

As mentioned by Hoffman [46], DPI is used by the so-called Golden Shield project, also known as The Great Firewall of China. This project aims to censor China’s Internet by several technical procedures. Specifically, DPI is used for detecting sensitive content in unencrypted packets — keywords in search engine queries and more [46].

1.2.2 IDS/IPS Systems

An intrusion Detection System (IDS) is a “software or an appliance that detects a threat, unauthorized or malicious network traffic” [47]. As also mentioned in [47], the purpose of IDS is to provide monitoring, auditing and reporting of the malicious activities on a network. Fuchsberger [48] described several types of IDS systems. Behavior-based IDS uses statistical techniques to detect anomalies and if a threshold is exceeded, an alert is generated. This could be for example a number of failed logins. On the contrary, Knowledge-based IDS looks for known attack patterns of the network traffic, such as byte sequences which are known to cause buffer overflow attacks. Host-based IDS typically runs on the sensitive hosts as an application and analyzes log files. The last described type by Fuchsberger [48] is Network-based IDS which analyzes network traffic on the packet level. Both packet headers and payloads are searched for attack signatures and generate alerts when a match is found.

As explained in [48], due to financial losses from downtime, information leaks and others, the

²www.suricata.io

³www.snort.org

market demanded systems that not only detected attacks but were also able to prevent them, known as the Intrusion Prevention Systems. Intrusion Prevention System (IPS) is “a product that focuses on identifying and blocking malicious network activity in real time” [48]. Fuchsberger [48] described two types of IPS. Rate-based IPS manipulates the network traffic based on load (too many packets, connections, and more). Moreover, it can be used to limit number of queries to DNS servers or to limit traffic traveling to a given port or a service. Content-based IPS (also known as a signature- and anomaly-based) manipulates traffic based on attack signatures. It can be used to block worms, packets that do not comply to the TCP/IP RFCs and suspicious behavior such as port scanning.

1.2.3 Flow-based network monitoring

Crotti et al. [49] state that mechanisms to classify network traffic based on full packet analysis are becoming ineffective. These mechanisms are too computationally demanding due to the increasing number of Internet users and services and can not be used on high-speed networks [49]. Flows and flow-based monitoring address this problem. Only the packet headers are analyzed, leaving the carried data untouched. A flow is defined as “a set of packets or frames passing an Observation Point in the network during a specific time interval” [50]. As mentioned earlier, a flow typically works only with packet headers, so it is less privacy-sensitive when compared to the classical DPI. Moreover, it significantly reduces the amount of data that needs to be processed and analyzed. Thus, it is more scalable in high-speed networks, but data can still easily exceed tens of terabytes [51].

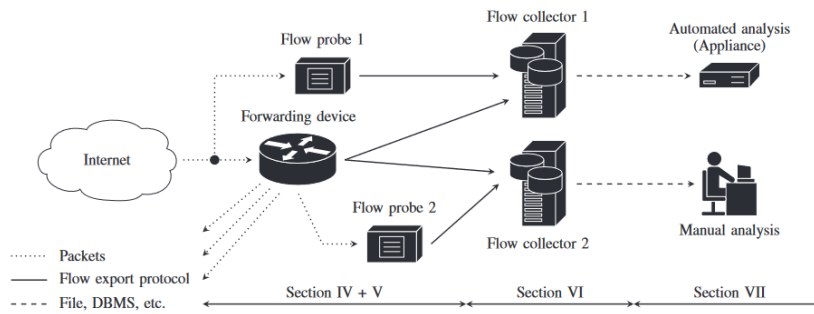
Packets are aggregated into flows by the flow key, which is a hash calculated from the following values:

1. Source IP address
2. Destination IP address
3. Source port
4. Destination port
5. Used protocol on the transport layer

According to Hofstede et al. [51], typical flow-based network monitoring setup has several stages (shown on figure 1.6). Firstly, packets are captured and pre-processed by the observation points. The second stage consists of aggregating packets into flows (metering process). After a flow is considered terminated, it is exported (exporting process) by a flow export protocol — added to a datagram of this protocol. Datagrams may include both flow characteristics (such as IP addresses, ports, ...) and measured properties (byte and packet counters, ...). Exported flows are then pre-processed (data compression, ...) and stored in the data collection stage. The last stage is data analysis which usually includes classification, anomaly detection, and more.

IPFIX is defined as “a unidirectional, transport-independent protocol with flexible data representation” [52] (flow export protocol) and is used for encapsulation and transportation of flow records.

A flow can be considered terminated because of three reasons. In the case that a flow has been active for a certain period of time, *active timeout* aims to help with such long-lived flows periodically. Typical active timeout values range from 120 seconds to 30 minutes [51]. Another reason is due to *passive timeout*. This means that there have not been any observed packets belonging to a flow and therefore it is considered terminated. Typical values are from 15 seconds to 5 minutes [51]. Lastly, *resource constraints* are defined, meaning that a flow can be considered terminated in advance to save resources.



■ **Figure 1.6** Flow-based network monitoring architecture, taken from [51]

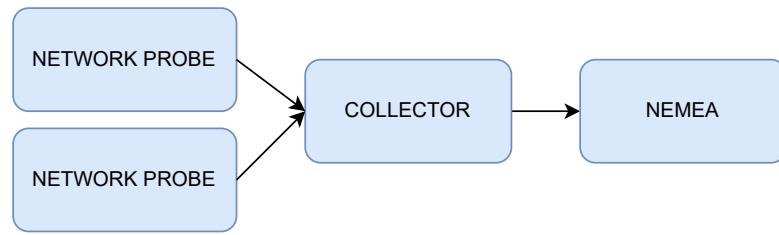
1.2.4 NEMEA

According to Čejka et al. [53], there are several methods for detecting malicious attacks based on the flow data. Nevertheless, since attacks are getting more sophisticated, flow data are not sufficient for detection. Headers of the application layer (L7) may be needed for reliable detection. Unfortunately, there is a lack of tools that supports parsing of L7 information. Network Measurements Analysis (NEMEA) is a platform for stream-wise traffic analysis and anomaly detection to overcome this problem.

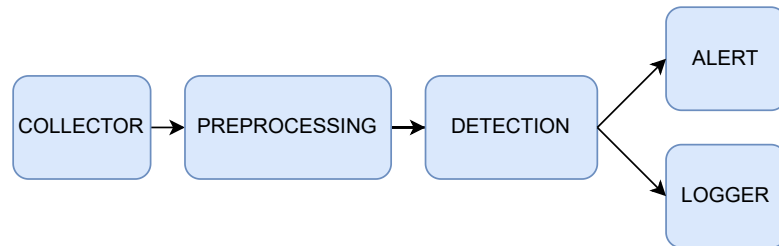
Čejka et al. [53] proposed NEMEA as a heterogeneous modular system where modules are chained together by unidirectional interfaces. These interfaces transfer data in streams of messages which can be flow records, analysis results, alerts and more. Each module performs a specific task such as flow data preprocessing, filtration, anomaly detection or logging and reporting results. Modules can be interconnected in various ways and NEMEA deployments can be composed of entirely different sets of modules performing needed tasks. NEMEA is therefore very flexible and offers a high level of customization.

A typical network monitoring setup, as proposed by Čejka et al. [53], is shown in figure 1.7. Monitoring probes capture packets, export flow data, and contain plugins for parsing L7 information and extending classical flow data. Flow data are then sent to the central collector, which stores and resends them to the NEMEA system for analysis. Module interconnection usually forms a directed acyclic graph or a tree, as shown in figure 1.8. A single module, the root of this tree, usually serves as an input to all other modules. It either creates or gathers flow records and sends them to its output for further processing. On the other side of this there are modules used for reporting. CESNET in its deployment uses a plugin for the IPFIXcol as the root. This module receives and parses IPFIX messages from the monitoring probes, translates them into the NEMEA message format and sends them to its output.

As mentioned above, NEMEA modules use a particular format for communication — UniRec, which was also described by Čejka et al. [53] It is a generic and efficient binary format for storage and transfer. It also supports variable-length fields and allows to define specific formats via templates. Furthermore, NEMEA supports JSON format and unstructured data. However, these two formats are not commonly used. The main advantage of UniRec against the other formats is that it allows very fast access to the flow fields. It does not require to be parsed and allows direct reading. Almost as fast as a plain C struct with the advantage that UniRec can be defined in runtime [53].



■ **Figure 1.7** Network monitoring setup with NEMEA



■ **Figure 1.8** Internal architecture of NEMEA

1.3 Machine Learning

Mahesh [54] described Machine Learning (ML) as “the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed”. ML uses different algorithms (models) to perform its tasks. The model is fit to the training data during the training phase and the model’s exact parameters are determined [55]. Trained algorithms can make predictions afterward on the previously unseen data. IBM Cloud Education [56] divides ML methods into three categories — *supervised*, *unsupervised* and *semi-supervised*.

Based on [56], supervised ML uses labeled datasets — every sample has a label containing its true class. Labels are used in a process called cross-validation to evaluate if a model is trained appropriately. These methods can be used to classify emails as spam and many other real-world problems. Linear and logistic regression or random forests are examples of such methods.

On the contrary, unsupervised ML uses algorithms to cluster and analyze unlabeled datasets [56]. Algorithms are able to discover hidden patterns or groups of data on their own, without the need of human touch. It is suitable for data analysis, image and pattern recognition. An example of such algorithm is the principal component analysis (PCA). PCA is able to reduce number of features in a model [57]. Other examples are K-means clustering or probabilistic clustering methods.

Semi-supervised ML was also described in [56]. It combines two previously described methods together. Small labeled dataset is used for feature selection during training. This is useful when we do not have enough labeled data for training a supervised ML algorithm. Moreover, IBM Cloud Education [56] mentions Reinforcement ML. There is no training phase and a model learns “as it goes by using trial and error” [56]. A deeper examination is however out of the scope of this thesis.

Ensemble learning is a method that uses several ML models and combines them together for better performance. Mahesh [54] described ensemble learning as “the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem”. Sagi et al. [58] explained that the base premise of

ensemble learning is that errors of a single model will likely be compensated by others and the overall prediction would be improved.

According to Pykes [59], ML algorithms can be used for solving two types of problems. Classification problems involve predicting labels (discrete values). On the other hand, regression problems predict quantities (continuous real numbers).

1.3.1 Selected supervised ML algorithms

K-nearest neighbors algorithm (KNN) was initially developed by Fix et al. [60]. KNN algorithm uses similarity and makes predictions based on the k -nearest neighbors [61]. The majority vote of neighbors is used to obtain the final prediction. The training phase only consists of storing features and labels.

Logistic regression uses independent variables as predictors of the dependent variable [62]. It is used to predict categorical dependent variables [63], which are in range between 0 and 1 [62]. On the contrary, linear regression is primarily used for regression problems [63] since it makes predictions on continuous dependent variables. Linear and logistic regressions both use dependent and independent variables [63] and are very similar.

Based on Gupta [64], Decision tree is a model which uses a tree for making predictions. Internal (non-leaf) nodes have input features and they represent a split. Edges coming from a node are labeled with each possible value of the input feature. Each leaf is labeled with a class or class probabilities. The model's tree is built by splitting the training dataset into subsets by several rules (which can be tuned by hyperparameters). This process is recursively repeated. Decision trees are favorite and widely used due to their explainability — a tree can be visualized and reviewed.

As described by Breiman [65], Random forest is an ensemble algorithm that uses multiple decision trees to significantly improve prediction accuracy by “letting them vote for the most popular class” [65]. Akar et al. [66] state that trees are grown using random feature selection and creates a new training dataset for each of them. Random forests are very fast, robust and achieve significantly better performance than a single decision tree [66].

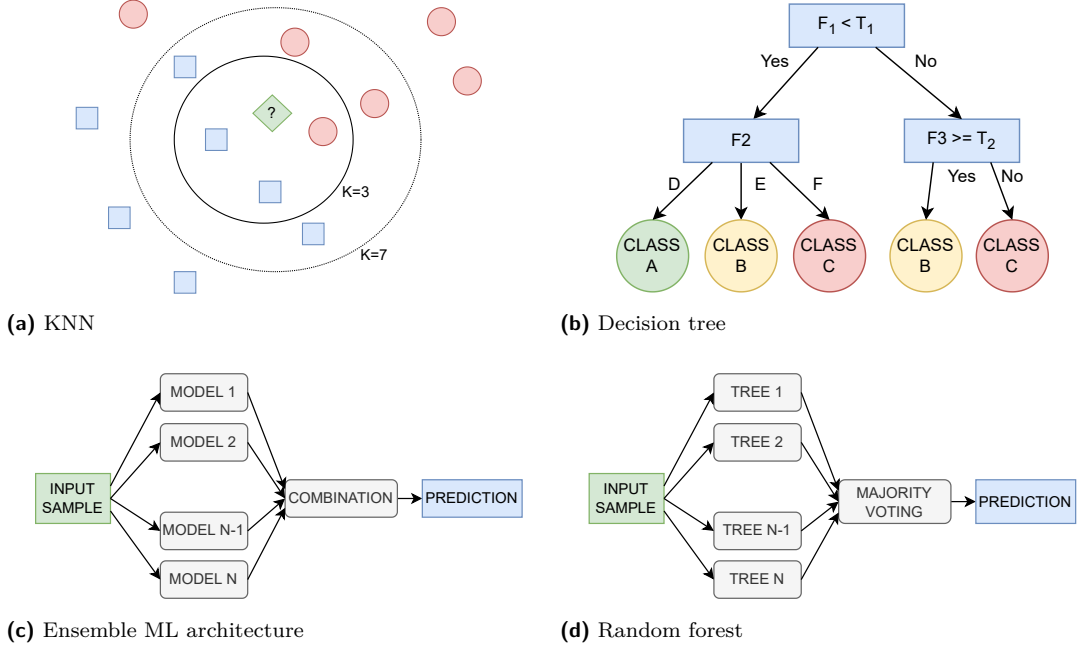
Adaptive Boost (AdaBoost) is another ensemble learning method. Schapire [67] explained that boosting uses a lot of weak and inaccurate rules, combines them and creates a highly accurate prediction rule. AdaBoost was the first practical algorithm to use boosting method [67].

Some of the models described here are shown on the figure 1.9.

1.4 Dempster-Shafer theory

Dempster-Shafer Theory (DST) is a mathematical theory of evidence. Original work [68] of Arthur P. Dempster was later expanded in [69] by Glenn Shafer. It is a generalization of the Bayesian theory of subjective probability. The DST is based on two ideas: the idea of obtaining degrees of belief for one question from subjective probabilities for a related question and Dempster's rule for combining such degrees of belief when they are based on independent items of evidence [70]. The Bayesian theory requires probabilities for each element, but the DST allows us to base degrees of beliefs for one question on other related questions. Degrees of beliefs defined this way may or may not have mathematical properties of probabilities. It depends on how the questions are related.

It was also presented in [71] as the “Evidence Theory”. Pieces of evidence can be seen as events that occurred or can occur in a system. Evidence implies a hypothesis or sets of hypotheses. The DST initially assumes a Frame of Discernment, shown in the equation 1.1, which is defined



■ **Figure 1.9** Selected ML models

as a set of primitive hypotheses. Another set is then formed by all subsets of Θ , creating the set containing all possible hypotheses (power of two), shown in the equation 1.2:

$$\Theta = \{h1, h2\} \quad (1.1)$$

$$2^\Theta = \{\emptyset, \{h1\}, \{h2\}, \{h1, h2\}\} \quad (1.2)$$

As described in [71], degrees of belief of elements of 2^Θ are represented by the mass function. The mass function indicates how strong a piece of evidence supports a hypothesis or how much an evidence agrees with a hypothesis. This is expressed by a number from 0 to 1. The DST requires that the sum of masses assigned to hypotheses is equal to 1.

Moreover, multiple sources of evidence can be combined together via the Dempster's Rule of Combination to produce a better estimate about hypotheses. It combines multiple mass functions and produces a new mass function that represents original hypotheses and possibly conflicting pieces of evidence. As described by Smets [72], another rule, the conjunctive combination rule, was derived from the original Dempster's Rule of Combination. This is used to compute degree of belief $bel(A \wedge B)$ from $bel(A)$ and $bel(B)$.

The transferable belief model (TBM) is Smets et al.'s [73] interpretation of the DST. TBM is a two-level model:

1. *Credal* level — beliefs are taken into consideration
2. *Pignistic* level — beliefs are used to make decisions

Belief functions represent beliefs at the credal level and are updated in time. On the contrary, only when a decision has to be made the pignistic level appears. When the pignistic level appears, belief functions are transformed via the *pignistic transformation* to probability functions which are used to make a decision. A real-world example usage can be found in [74] and [75], however further study of this topic is out of the scope of this thesis.

Datasets

To be able to design a detector, we needed a sufficient amount of example data of both miner and non-miner traffic. We decided to collect traffic on the CESNET2 network (operated by CESNET) to get as much real-world data as possible. This chapter describes how we selected cryptocurrencies for further investigation and how the traffic was captured.

2.1 Current situation of cryptocurrencies

As mentioned by Chatzigiannis et al. [76], it is almost impossible for solo miners to compete in the mining process, even with specialized hardware. They also state that the vast majority of miners is connected in mining pools, which was also our assumption. Therefore, we will only focus on pooled mining. As the first step, we decided to inspect current situation of cryptocurrencies and mining pools and select a couple of them for our follow up work. We chose Bitcoin (BTC) because it is the most famous cryptocurrency and has currently the biggest market cap [77] (total value of all mined coins [78]). The second selected cryptocurrency is Ethereum (ETH). It is also a very famous cryptocurrency and has the second biggest market cap [77]. In addition, we selected Monero (XMR) because a significant amount of coins is mined by malware (as described in subsection 1.1.3). Overview of the selected cryptocurrencies:

1. Bitcoin (BTC) — well-known crypto, the biggest market cap
2. Ethereum (ETH) — well-known crypto, the 2nd biggest market cap
3. Monero (XMR) — significant % of mined coins comes from malware

We also chose the top 10 mining pools by the overall hash rate for each selected cryptocurrency. Source for BTC pools is [79], ETH pools are based on [80] and XMR pools come from [81]. Furthermore, we extracted mining protocols and opened ports for mining, so we can later use this information for traffic capture. Some pools required registration or did not allow the public to mine at all. We still decided to include them in our list to demonstrate the percentage of “private” pools in the top 10. Complete overview of the selected mining pools, mining protocols and ports is available in the appendix A.

After inspection of the extracted data, we can state that every pool has either directly specified Stratum as the used mining protocol (marked as “Stratum” in the appendix A) or mentions Stratum in a “Help” section with example configurations of the mining software (marked as “Stratum?”). Apart from Stratum, SlushPool also supports its own and newly created protocol

Stratum V2. Moreover, one pool mining ETH also supports the Getwork protocol but simultaneously supports Stratum as well. We suspect this is for legacy reasons. The majority of mining pools also supports connection via TLS/SSL. A significant amount of miner traffic may be encrypted and protected against the DPI-based detection methods.

2.2 Local examination of miners' traffic

Before capturing traffic on the CESNET network, we decided to examine the traffic generated by a miner run locally in a virtual machine. Firstly, we prepared the virtual machine with Debian 11, 8 GB RAM, 8 processor cores and installed XMRig (miner software¹ for Monero). We used Wireshark² for traffic capture and inspection. Then we let the miner run for several hours. We also connected to the three different mining pools to see if there was any difference. Every mining pool we connected to used Stratum as the mining protocol.

Based on our few observations, we can state that miner's communication is long-lasting and undisturbed. Packets of small sizes are transferred in periodical time intervals, in our case usually between 30 and 70 seconds. However, we suspect it can differ based on the used mining software and the computational power of hardware it is running on. Most packets had the TCP PUSH flag set. Moreover, number of request and response packets was almost balanced. However, due to possible network problems and notifications (defined in 1.1.2) it was not precisely 50%. Finally, we did not find any visible difference between traffic based on a mining pool the miner connected to.

We suspect that traffic generated by instant messaging applications (such as Facebook Messenger and others) or generic checking for updates can have similar characteristics. This observation is also based on the previous work (described in subsection 1.1.4). This could potentially cause false predictions and increase the number of false positives.

2.3 Traffic capture on CESNET

After the initial inspection of miners' traffic (described in section 2.2), we decided to capture traffic on the CESNET network. We used the information extracted from mining pool websites (described in section 2.1) to create a rule to filter traffic. We resolved mining pool FQDNs to both IPv4 and IPv6 addresses and paired them with corresponding ports. This produced pairs of IP addresses and ports which were then concatenated into a rule. This rule was then used for traffic capture.

However, this rule will not cover potential changes in mining pools' architectures. As described by Žádník et al. [27], mining service may be available on the new servers to improve load balancing. Moreover, servers can be rented in the cloud and cloud providers often rotate IP addresses [27] and therefore mining pool servers hosted in such way can often have different IP addresses over time. A possible solution was to use active probing for each flow to determine if one of the IP addresses belongs to a mining pool. However, that would be slow as described in subsection 1.1.4 and we could be potentially marked as a scanner and blacklisted on the network borders. In addition, it raises an ethical question. We decided to collect several mining pool lists online, merge them together and run a script (described below) periodically to re-generate the traffic capture rule once per day.

We created a simple Python script, which takes a list of mining pool FQDNs and ports used for mining by each pool. Firstly, FQDNs are resolved to IP addresses (both IPv4 and IPv6). Pairs containing all combinations of the pool's IP addresses and ports are then generated by

¹www.xmrig.com

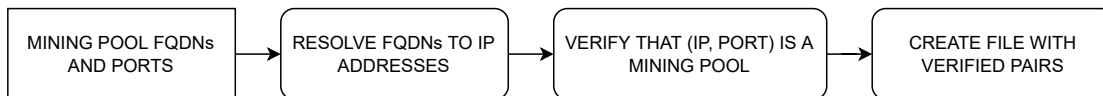
²www.wireshark.org


```

{
  "id":1,
  "jsonrpc":"2.0",
  "method":"login",
  "params": {
    "login":"45pw...Ldc"
  }
}

```

■ **Code listing 2** Stratum request used in active probing, value of *login* is shortened



■ **Figure 2.1** Scheme of the Python script for rule generation

the Cartesian product (shown in the equation equation 2.1). Active probing is then used for verification that there really is an up and running mining pool on `IP:port`. Stratum request (shown on listing 2) is sent and obtained response is parsed as a JSON. However, it is possible for a server to return JSON even without being a mining pool, so we logged all the responses and reviewed them manually. Because no errors were detected and all successfully parsed JSONs contained Stratum, we automated this process so the rules are generated without the need of manual review. If a server from the input list returned JSON for a Stratum request, we considered this enough for the server to be a mining pool. The final stage writes verified pairs to a file. This file has one pair per line, with the following structure: `IP port\n`. Proposed scheme is shown on the figure 2.1. As mentioned above, this script was set to run once per day to re-generate the output file with a CRON job. The structure of the output file is made to be compatible with another module — *blacklist*. Our output file was used as a filter for this blacklist module for traffic capture.

$$\{\text{IP addresses}\} \times \{\text{Ports}\} = \{(\text{IP address}, \text{Port})\} \quad (2.1)$$

We noticed that ports used by mining pools are usually very similar and therefore we propose an improvement for our script. Instead of relying on the input mining pool list that no other ports are used, we can scan the input mining pool list and create a set of all “discovered ports”. Later, when generating pairs, we may not only combine IP addresses and their corresponding ports but create a set of pairs with all discovered ports. This way we can scan mining pools for known ports used for mining and not only for those obtained from the input mining pool list for a specific pool. However, we decided not to implement this feature because it would be time consuming and even without this improvement the blacklist was capturing a sufficient amount of miners’ traffic. Moreover, we would scan ports we know very little about and we could be marked as a scanner (described above).

As the opposite class of traffic (non-miner) we decided to use the traffic from one of the CESNET’s subnets. Traffic on this subnet should contain all the types, such as the traffic generated by Internet browsing, streaming services, Voice over IP. Moreover, traffic generated by instant messaging and update checks should be included as well. Therefore, it is a good counter-class for miners’ communication since it contains the majority of other types and possibly similar communication as well (discussed in section 2.2).

```
/usr/bin/nemea/logger -t -i "f:$FILE" -w "$FILE.csv"
```

■ **Code listing 3** Bash command used for conversion of a trapcap file into csv

■ **Table 2.1** Datasets overview, Finalized Dataset (FD) is a dataset created by appending all four datasets together

Name	Time range	# Miner flows	# Other flows	# Total flows
01	Dec 14-17 2021	195 646	343 879	539 525
02	Jan 01-17 2022	239 990	472 139	712 129
03	Jan 17-31 2022	142 844	322 399	465 243
04	Feb 01-10 2022	114 757	193 249	308 006
FD	—	693 237	1 331 666	2 024 903

2.4 Creating datasets

Traffic captured by the blacklist module and our rules (described in section 2.3) was saved in the UniRec format. We used NEMEA’s module called *logger*³ to convert these trapcap files into CSV files (the exact command is shown on listing 3). Data in files with traffic based on our rules were labeled as *Miner*. Data from the CESNET’s subnet (also discussed in section 2.3) was labeled as *Other*. Because the non-miner data could be potentially miners’ traffic, we merged IP addresses and ports from our rules generated over time and ran a check. However, we did not find any flows that would satisfy the rules for capturing miners’ traffic in the non-miner data. Unfortunately, there can still be miners’ traffic present in non-miner data because our input mining pool lists are not complete.

As it was pointed out to us, anomalies can cause wrong measurements and insufficient data in some of the exported flows, we decided to filter obtained flows before further work. Some flows represented communication in only one way or had zero bytes or packets. We decided to filter out these flows because they bear almost no information and therefore are not suitable for detection. A flow was dropped if one of the following criteria were met:

1. PPLPKT_DIRECTIONS contained packets in only one direction (all 1 or all -1)
2. BYTES == 0 or BYTES_REV == 0
3. PACKETS == 0 or PACKETS_REV == 0

To sum this up, we created four datasets containing both miner and non-miner flows. Each dataset contains data from approximately half of a month. This is based on when we actually got the data. Moreover, it may be used to examine if and how miners’ characteristics changed over time. Overview of created datasets can be found in the table 2.1. Datasets were also anonymized, meaning that IP addresses were replaced by their hashed variants. Anonymized datasets are available on the attached medium. In addition, the finalized dataset was created by appending all the datasets together (used later). This finalized dataset is not available on the attached medium since it can be easily created from the datasets 01-04.

2.5 Summary

In this chapter, we inspected the current situation of cryptocurrencies and mining pools. We chose BTC, ETH, XMR and for each cryptocurrency also the top 10 mining pools by the overall hash rate. We examined locally captured communication of the XMRig miner and created a solution for automatic capturing traffic of miners' communication on the CESNET network.

Traffic capturing took place from about mid of December 2021 until mid February 2022. Specifically, miners' communication was captured during December 14th, 2021 and February 10th, 2022. The non-miner traffic was captured during December 17th, 2021 and February 10th, 2022. Data were filtered and labeled. Additionally, we split the data into several datasets based on days based on dates we got hold of the data. Filtered datasets with anonymized IP addresses and without IDP CONTENT and IDP CONTENT REV are available on the attached medium.

³www.github.com/CESNET/Nemea-Modules/tree/master/logger

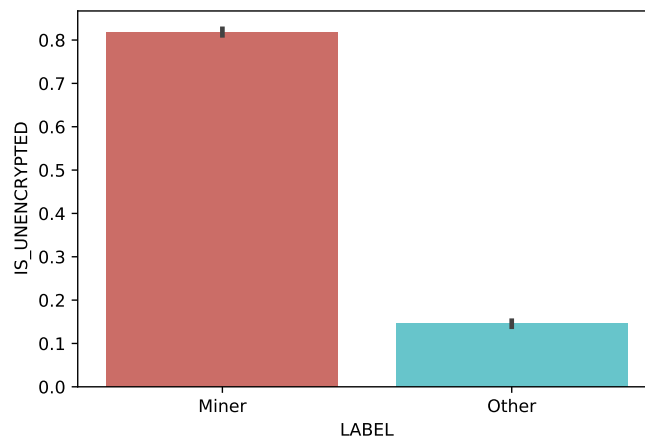
Analysis and design

Traffic captured on the CESNET is thoroughly analyzed in this chapter and the detector's complex design is proposed based on this analysis. Then, several modular parts of our detector are described together with their final combination. One section provides the implementation details. The implementation process showed that two new modules are needed for the successful deployment of our module. These modules are also described in this chapter.

3.1 Analysis of traffic from CESNET

We used the finalized dataset (described in the section 2.3) for our experiments. All traffic was analyzed, however a random sample with a total of 20 000 flows (10 000 flows per class) was used for plotting figures shown in this section. A complete overview of flow data measured and exported on CESNET network and therefore features for potential distinguishing of flows is shown in the table 3.1.

Firstly, we examined the usage of encryption. The figure 3.1 shows how many percentage of flows of each class bear encrypted communication. It is clear that many miner flows do not use encryption and therefore are vulnerable to the DPI-based techniques.

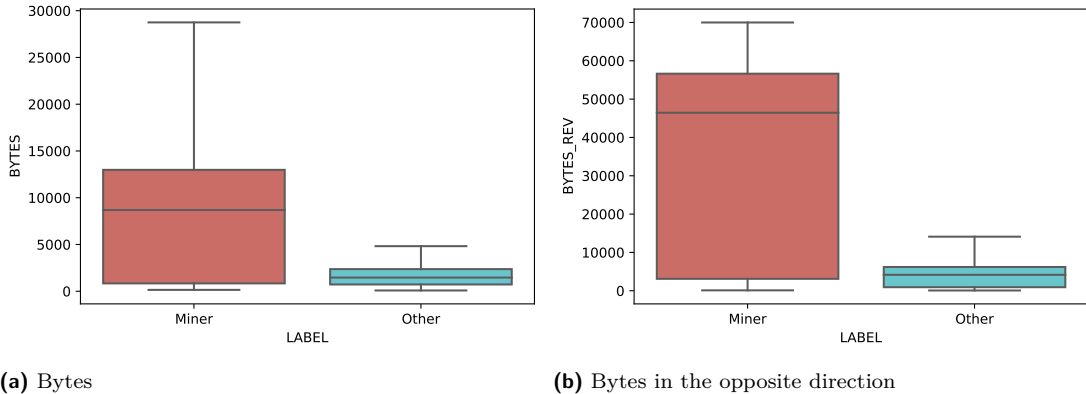


■ **Figure 3.1** Unencrypted flows in CESNET traffic

■ **Table 3.1** Overview of flows' metrics exported on CESNET network available for detector

Name	Description
SRC_IP	Source IP Address
SRC_PORT	Source Port
DST_IP	Destination IP Address
DST_PORT	Destination Port
PROTOCOL	Used protocol on the transport layer
LINK_BIT_FIELD	Exporter ID from where this flow came
TIME_FIRST	Timestamp of the first packet
TIME_LAST	Timestamp of the last packet
BYTES	Transmitted bytes from source to destination
BYTES_REV	Transmitted bytes in the opposite direction
PACKETS	Transmitted packets from source to destination
PACKETS_REV	Transmitted packets in the opposite direction
TCP_FLAGS	TCP flags of the first packet
TCP_FLAGS_REV	TCP flags of the first packet in the opposite direction
IDP_CONTENT_REV	First 100 bytes from destination
IDP_CONTENT	First 100 bytes from source
TLS_JA3_FINGERPRINT	JA3 Fingerprint
TLS_SNI	TLS SNI value
PPLPKT_DIRECTIONS	Directions of the first 30 packets
PPLPKT_FLAGS	TCP flags of the first 30 packets
PPLPKT_LENGTHS	Packet lengths of the first 30 packets

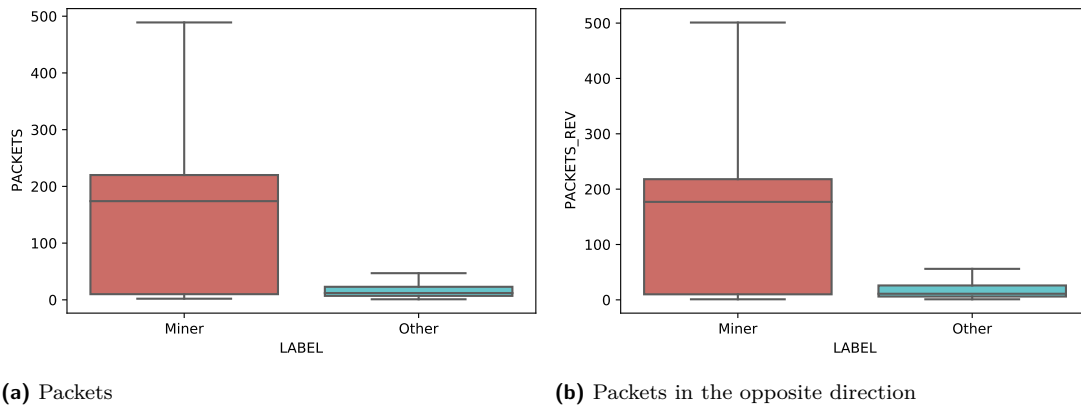
Next, we examined how many bytes and packets are transmitted between a miner and a mining pool. Numbers of exchanged packets in both directions are very low, as shown on the figures 3.3a and 3.3b. Comparison of transmitted bytes is shown on the figures 3.2a and 3.2b.



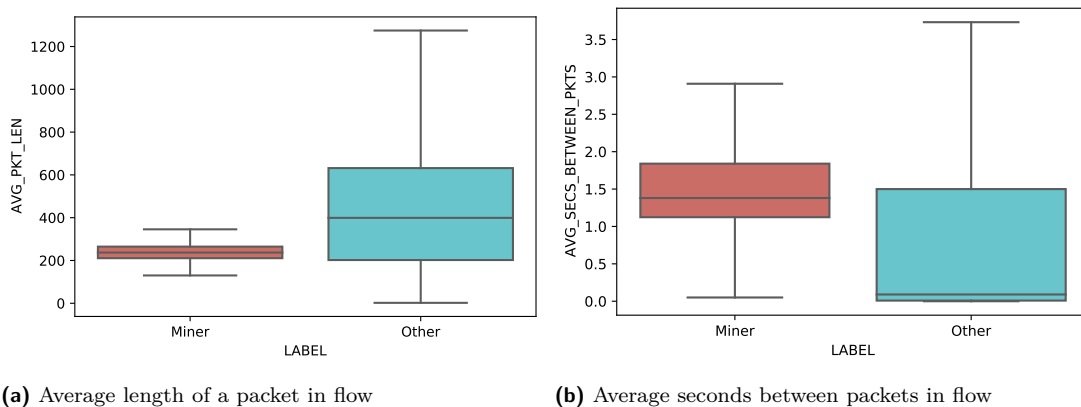
■ **Figure 3.2** Statistics of exchanged bytes

Moreover, we decided to examine how long is an average packet in a flow and how many seconds pass between each packet in a flow. As shown on the figure 3.4a, the majority of miner flows have an average packet length below 600 bytes. As we can see on the figure 3.4b, miner flows have longer time gaps between packets in general.

We also calculated the ratio of sent and received packets in a flow and looked for any significant difference. As we can see on the figure 3.5a and figure 3.5b, non-miner flows ratios range through



■ **Figure 3.3** Statistics of exchanged packets



■ **Figure 3.4** Packet characteristics

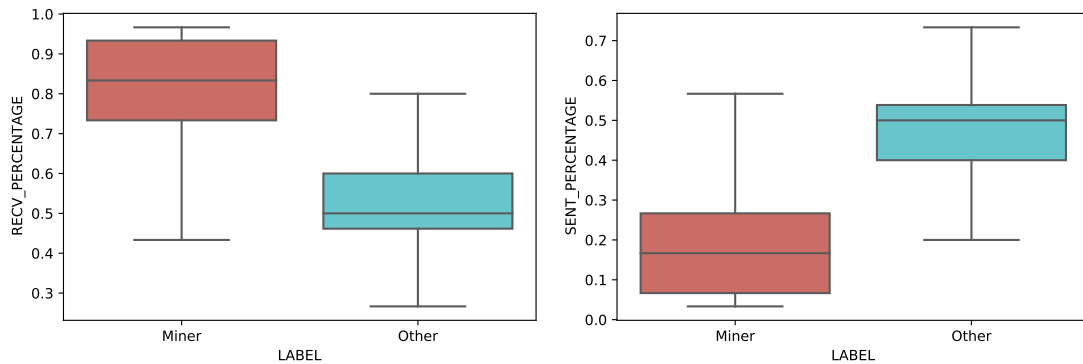
the whole spectrum. On the other hand, miner flows have more received packets. The graphs showing sent and received packet ratios of miner flows look like expected, based on the Stratum specification.

Data showed on the figure 3.6 also confirmed that miner flows have packets in a flow with TCP PUSH flag set is high, from 70% and more. We also calculated and inspected ratios of other TCP flags such as ACK, FIN or RST. However, we later determined that these ratios bear very little information which could be used to distinguish miner flows.

Lastly, we calculated the overall flow durations, shown on the figure 3.7. Most miner flows are either below 50 seconds long or more than 300 seconds long. On the other hand, non-miner flows have more equal scatter. Overall duration is based on the flow start and end times and not on the actual length of a mining session. Therefore, active and passive timeouts can affect these values.

3.2 Design

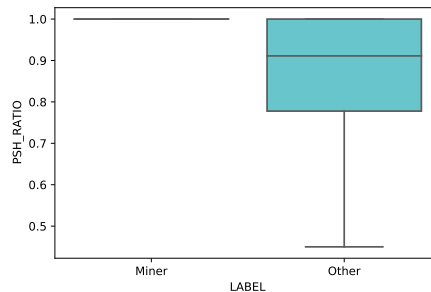
Our initial thought was to use several support detectors for classifying a flow as the miner. We decided to use indicators from different sources and then combine them together to get the final prediction. This idea was based on the (heterogeneous) ensemble ML where several classifiers are used when making predictions. The overall prediction is obtained by a combination of the



(a) Ratios of received packets in flow

(b) Ratios of sent packets in flow

■ **Figure 3.5** Ratios of received and sent packets in flows



■ **Figure 3.6** Ratios of TCP PUSH flag in flows

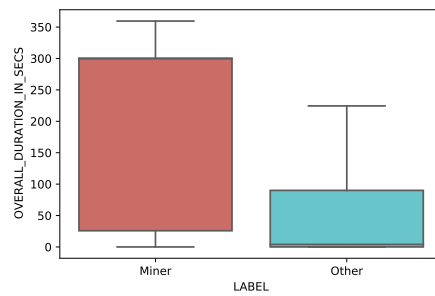
base classifiers' results which is more robust. The number of used models compensates for errors of every single one of them if they would be used alone.

We decided to use ML over the basic flow's characteristics that can be calculated for any flow — both encrypted and unencrypted. The advantage of this approach is that it can be used for processing any flow but can produce a lot of false predictions.

Since a large percentage of flows were unencrypted, we explored the possibilities of Stratum detection. CESNET exports the first 100 bytes from each direction, therefore 200 bytes are available for each flow. This detection is however only available for unencrypted flows. On the other hand, detection of Stratum will provide results with a very strong degree of belief.

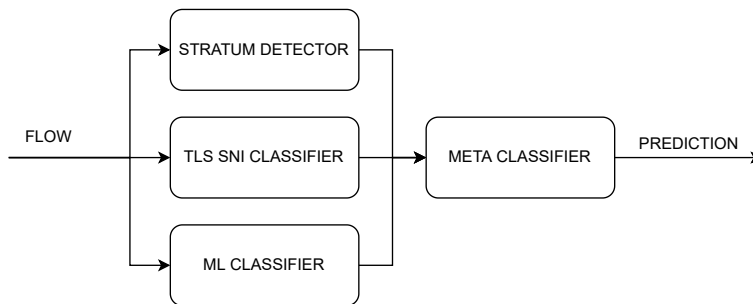
To provide a source also for the encrypted flows, we decided to use the SNI extension of the TLS protocol, which is available in the CESNET's setup. We can use the TLS SNI to detect suspicious keywords in the hostnames. The disadvantage of this approach is that the TLS SNI value is only present in the TLS handshake and thus during the initiation of the encrypted connection. Therefore, if there is a miner with a long-lived connection using the TLS, only the first flow will have the TLS SNI set. Following flows (created by timeout values of a network probe) will have the TLS SNI empty.

These three support classifiers are combined together by the Meta classifier. Firstly, the Stratum detector is invoked. If Stratum is successfully detected, flow is marked as miner right away since this is a very trustworthy indicator. Otherwise, the Meta classifier continues the process. ML classifier is invoked to get the probability of flow being a miner. Then, if TLS SNI is empty, the Meta classifier bases its decision only on the ML classifier. Otherwise, the TLS SNI classifier is



■ **Figure 3.7** Overall flows duration

invoked to obtain the TLS SNI score. Probability from the ML classifier is combined together with the TLS SNI score for the final prediction. Our proposed scheme is shown on the figure 3.8.



■ **Figure 3.8** High-level concept of the Meta classifier

Initially, we also proposed to use DNS as a support classifier. We would track IP addresses from flows generated by the DNS requests. However, DNS requests might be encrypted since DNS over TLS or HTTPS (section 1.1.2) can be used. Moreover, there is no guarantee that the source IP address of a DNS request corresponds to a machine that originated the DNS request. A client usually sends a request to a local DNS server, which communicates with other DNS servers to get the answer, which is then returned to the client. Due to this, we repudiated the DNS as the potential indicator.

3.3 ML classifier

The first classifier is based on the ML’s ability to predict class or class probability for previously unseen data. At first, we experimented with AdaBoost (with underlying decision tree), Decision tree, Logistic Regression and KNN. We used GridSearchCV¹ to find best hyperparameters of each model. We also used multiple input sets of hyperparameters based on several approaches — define hyperparameters via the constants or percentages of data.

We used *BYTES*, *BYTES_REV*, *PACKETS* and *PACKETS_REV* as features since miner flows are long-lasting and small amount of data is transmitted. We also calculated new statistics and then used them as features. *SENT_PERCENTAGE* and *RECV_PERCENTAGE* features are used to as ratios of transmitted data in each direction. The feature *IS_REQUEST_RESPONSE* is true if both ratios are equal to 50%. Otherwise it is set to false. The average number of seconds between packets in a flow is represented by *AVG_SECS_BETWEEN_PKT*. Flow’s overall direction in seconds is represented by *OVERALL_DURATION_IN_SECS*. *AVG_PKT_LEN*

¹www.scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

■ **Table 3.2** Overview of selected features for ML models

Feature name	Source	Description
BYTES	Flow	Number of bytes from src
BYTES_REV	Flow	Number of bytes from dst
PACKETS	Flow	Number of packets from src
PACKETS_REV	Flow	Number of packets from dst
SENT_PERCENTAGE	Calculated	Ratio of packets from src
RECV_PERCENTAGE	Calculated	Ratio of packets from dst
IS_REQUEST_RESPONSE	Calculated	True/False if sent packets are balanced
AVG_PKT_LEN	Calculated	Average length of a packet
AVG_SECS_BETWEEN_PKTS	Calculated	Average time seconds between packets
OVERALL_DURATION_IN_SECS	Calculated	Overall flow duration
PSH_RATIO	Calculated	Ratio of packets with PUSH flag set

■ **Table 3.3** Hyperparameters of the best Random forest model

Hyperparameter	Value
Criterion	<i>gini</i>
Max depth	10
Max features	<i>sqrt</i>
Min samples leaf	2
Min samples split	5
Estimators	100

represents the average packet length in flow and *PSH_RATIO* is the ratio of packets in flow that had TCP PUSH flag set. Overview of selected features is shown in the table 3.2.

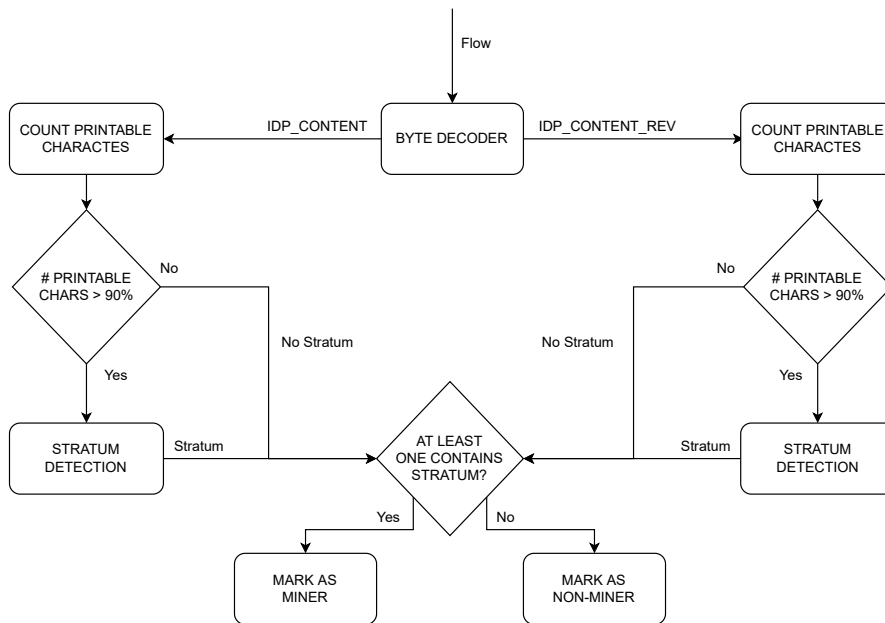
Decision tree achieved sufficient accuracy of 99.50% on both training and evaluation datasets and it also outperformed other models. Moreover, Decision trees are simple and explainable ML models, thus were selected for further tuning. Together with the Decision tree, we also experimented with Random forest since it is more robust and achieves better performance than a single decision tree. When finding hyperparameters, we used datasets 01 and 02. However, evaluation and model selection is based on all four datasets. Hyperparameters of the best Random forest model are shown on table 3.3, results on table 4.1.

3.4 Stratum detector

Our next detector is based on the pattern matching DPI technique and inspects packets' contents. Initially, we designed this detector as shown on the figure 3.9. Detector processes each value of the IDP CONTENT and IDP CONTENT REV independently. Firstly, the percentage of printable characters is calculated. Stratum detection is done if and only this percentage is greater than 90%, meaning that a flow bears unencrypted communication. A flow is marked as a miner if at least one of the values contains Stratum.

Ipfixprobe² deployed in the CESNET monitoring infrastructure exports first 100 bytes from each direction for each flow, so we decided to look for typical Stratum strings in these bytes. It is important that only parts of the Stratum strings are available and may look like invalid JSON. Therefore, it is impossible to use a JSON parser. Moreover, JSON carrying Stratum request (and notification) has different keys than Stratum response. We propose two groups of keywords

²www.github.com/CESNET/ipfixprobe



■ **Figure 3.9** Initial design of the Stratum detector

for Regex matching of each of the types, based on the Stratum specification [28] and our own observations in the captured data. We also match the enclosing quotes and colon to ensure it is part of a JSON structure and not the regular text.

Regex patterns for Stratum request and notification:

1. "method"\s?: — Stratum specification
2. "params"\s?: — Stratum specification
3. "jsonrpc"\s?:"2.0" — Our observations
4. "worker"\s?: — Our observations
5. "mining\.set" — Our observations
6. "mining\.not" — Our observations

Regex patterns for Stratum response:

1. "id"\s?:
2. "result"\s?:
3. "error"\s?:

The Regex rule for matching “id” in Stratum response produced a lot of false positives. After inspection, we discovered that all three matched strings are present. Therefore we decided that all three keywords have to be matched. The proposed Regex pattern matches three or more subgroups and the presence of all three keywords is checked afterward directly in the code.

We were able to match all flows containing Stratum in our datasets with the Regex patterns above. For optimizations, we created one compiled Regex for each group so that the Regex module will run only twice instead of nine times. Used patterns are on listing 4 and listing 5.

```
("(jsonrpc|method|worker)":\s?")|(params":|mining\.(set|not))
```

■ **Code listing 4** Complete Regex pattern for matching Stratum request and notification

```
((("(?P<I>id)|(?P<R>result)|(?P<E>error)":\s?).*){3,}
```

■ **Code listing 5** Complete Regex pattern for matching Stratum response

3.5 TLS SNI Classifier

TLS SNI classifier inspects the TLS SNI value. Since that SNI contains a website's hostname or domain name, we can use it to detect suspicious keywords in that hostname. We noticed that many mining pool hostnames contain words associated with mining, such as ether**mine**.org or beep**pool**.com. Moreover, they often use subdomains for mining of a specific cryptocurrency and setting up servers for different countries or continents, such as **xmr-eu1**.nanopool.org or **eth-us-west**.flexpool.io. Based on these observations, we designed our classifier to look for keywords from two groups — cryptocurrency names and mining keywords.

The first group of keywords is made out of short names of cryptocurrencies. Originally, we were matching *btc*, *eth* and *xmr*, case-insensitively. But since these short names are made from three letters, it was producing some false positives. To keep the false-positive rate as low as possible, we once again inspected the TLS SNI values and discovered that short names are usually pre-fixed or post-fixed by either “.” or “-”. Based on this discovery, we made several rules which transform each short name into the four enhanced patterns. Classifier takes the list of the short names on input and creates a new list of enhanced patterns for later matching. Each enhancement rule takes a short name as an input (marked as \$NAME) and produces a new pattern:

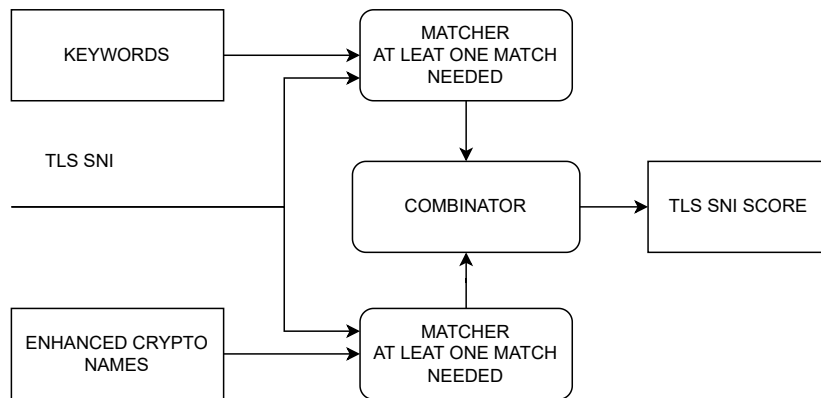
- -\$NAME
- \$NAME-
- .\$NAME
- \$NAME.

The second keywords group contains simple words indicating the mining process. Selected keywords are based on the values found in the TLS SNI in the captured data from the CESNET network. Matching is done case-insensitively. Selected keywords are:

- Pool
- Mine
- Mining

Finally, results from both parts are combined together. For a group to be matched, at least one keyword from that group has to be present in the TLS SNI value. For a full match, both groups have to be matched. For a partial match, at least one group has to be matched. TLS SNI score is then assigned based on the following rules:

score = 1.0, if full match
score = 0.5, if partial match
score = 0.0, if no match



■ **Figure 3.10** Design of TLS SNI classifier

```
pattern = re2.compile("|".join(keywords))
```

■ **Code listing 6** Concatenation of keywords to create one Regex pattern

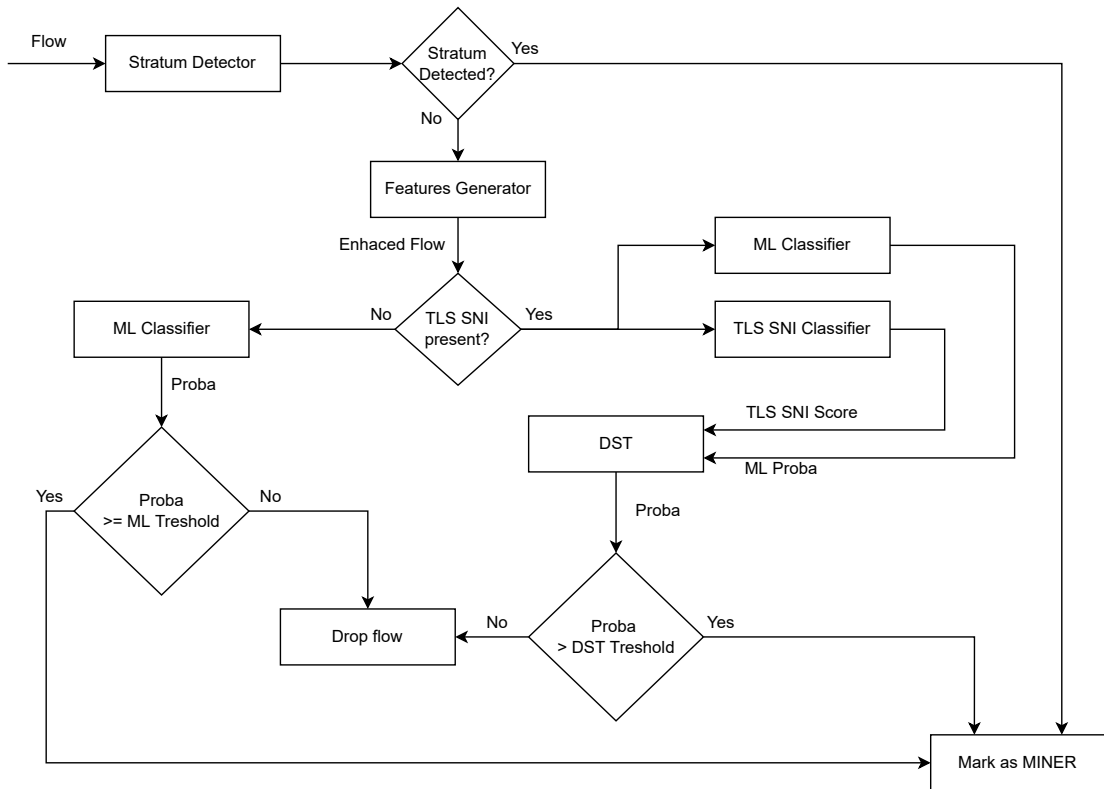
We only match a few short crypto names based on our data and even fewer keywords for mining pool hostnames. Potentially, a lot of false negatives can be produced. For example, pools mining crypto which is not in the list or a pool that does not have any reference to a mining process in its name. However, many cryptocurrencies are mined and to include all of them would be impossible. To keep our lists short and simple, we decided to keep only the names of cryptocurrencies that were actually present in our datasets, hence are mined in the CESNET network — BTC, ETH, XMR and RVN (Ravencoin). Since the TLS SNI classifier takes both keyword lists as an input, anyone can create a list that suits his needs. Many lists can also be found online.

As mentioned above, the TLS SNI classifier creates an enhanced list of crypto names for matching. Moreover, it generates one Regex pattern for each group by concatenating keywords by the OR operator, shown on listing 6.

3.6 Meta classifier

Meta classifier connects together support classifiers and uses their outputs to make the final prediction. Since detection of a mining protocol creates a very strong belief that source flow is a miner, successful detection of Stratum results in the Meta classifier marking this flow as the miner right away. Otherwise, we calculate features needed by ML and predict the probability of the flow being a miner. If the TLS SNI is not present, we compare this probability with the *ML threshold* to decide if the flow is a miner. If TLS SNI is present, we pass it to the TLS SNI classifier to obtain the TLS SNI score. TLS SNI score and ML probability are combined together via the DST (described in section 1.4) and compared with the *DST threshold* to make a decision. Schema of the Meta classifier is shown on figure 3.11.

We also considered the majority voting when combining support classifiers' decisions. However, the Stratum detector cannot provide a true prediction if a flow represents an encrypted connection. TLS SNI value is only sent during the initiation of the encrypted connection, therefore many flows representing the same connection initiated previously will have the TLS SNI empty. Moreover, the Stratum detector and TLS SNI classifier are mutually exclusive — a flow can bear unencrypted data or have a TLS SNI value, but not both at the same time. Therefore, the



■ **Figure 3.11** Schema of the Meta classifier

majority vote would be reduced to either ML and Stratum, or ML and TLS SNI. Since combining Stratum and ML with both DST and majority (logical *and*) only reduced the number of correct predictions, results of the Stratum detector are not combined but used directly. We decided to use the DST for a combination of the ML probability and TLS SNI score. DST allows us to express the trustworthiness of data sources numerically. On the other hand, logical *and* sees combining values as equals.

Initially, we designed Meta classifier to use the DST for combining results from all support classifiers. Moreover, Stratum and TLS SNI classifiers also provided the credibility values apart from scores. Credibility values were used to express belief in the corresponding classifier's output. For example, if the TLS SNI classifier detected a partial match and set the TLS SNI score to 0.5, credibility was set to 0.6 to express incompleteness of the keywords lists. These values were set experimentally. The credibility of the Stratum classifier was affected by a number of printable characters. The payload was considered text if the percentage of the printable characters was more than 90%. See complete overview in the appendix D. However, we later determined that this approach does not improve overall results.

TLS SNI score and ML probability are used to define a mass function for each value. M argument is used for miners, O represents others (non-miners). Definitions are shown in equations 3.1 and 3.2. These two mass functions are then conjunctively combined together and transformation is

```
pign = mlBpa.combine_conjunctive(tlsSniBpa).pignistic()
```

■ **Code listing 7** Conjunctive combination of mass functions to get pignistic function

done to get the resulting pignistic function, shown on listing 7.

$$\begin{aligned} bpa_{ML}(M) &= \text{ML probability} \\ bpa_{ML}(O) &= 1 - \text{ML probability} \end{aligned} \quad (3.1)$$

$$\begin{aligned} bpa_{SNI}(M) &= \text{TLS SNI SCORE} \\ bpa_{SNI}(O) &= 1 - \text{TLS SNI SCORE} \end{aligned} \quad (3.2)$$

The number of false positives and negatives is dependent on the values of DST and ML thresholds. To find the optimal DST threshold, we calculated pignistic functions to get miner probabilities from DST. Initially, we set the DST threshold to the value of 1% quantile from each of our datasets. ML threshold was set to 0.5. During the discussion with the thesis’s supervisor, we agreed that a large number of FP alerts could be counterproductive and we decided to look for the thresholds which would generate the lowest possible number of FP. We found the new DST threshold — 0.4419. Meta classifier still generated several FPs, but we determined that this was caused by the ML threshold. As described before, a number of false-positive alerts may be counterproductive — network administrators might stop paying attention to such alerts. To overcome this situation, the ML threshold was set to 0.9970. We also decided that miner detection will only take place if a flow has more than seven packets in each direction.

3.7 Implementation

We implemented the approach described above in a new NEMEA module called *miner detector*, written in Python. The first version was processing one flow at the time. Individual calls of Scikit-learn library³ caused this implementation to be very slow, around 260 flows per second. Therefore, we used a buffer to store flows until their count reached a certain number (can be set by an input parameter, default is 100 000) and then we processed the flows in bulk. Moreover, we used Cython⁴ to increase the overall performance and re2⁵ library for Regex matching. This improved the performance, but processed flows per second were still low. We also discovered an unnecessary parsing of datetimes when generating features. Fix of feature generation significantly improved performance.

It is possible that the ML model will worsen in time or that size of the buffer will not be suitable. Our module allows to set the ML model’s input path, buffer size and IFC interfaces via its arguments for users to customize it. Currently, keywords used in the TLS SNI classifier are hardcoded and therefore it is only possible to change them in code. However, it can be easily fixed and we plan to add this functionality in the near future.

We also discovered that the calculation of the percentage of printable characters in IDP CONTENT and IDP CONTENT REV does not produce better predictions. Our idea was that if a flow bear unencrypted communication and Stratum were not detected, we could mark it as a non-miner flow without running any additional support classifiers. However, it was only slowing

³www.scikit-learn.org

⁴www.cython.org/

⁵www.github.com/google/re2



■ **Figure 3.12** Proposed deployment architecture of our miner detector

down the process. Therefore, we removed this part and the Stratum detector does the detection directly.

During implementation, we also found out that raw outputs of the *miner detector* (flows marked as miners) can not be used for reporting purposes. As mentioned earlier, one connection can be represented by several flows over time, especially when this connection is long-lasting. Many raw outputs can therefore represent one active miner. Moreover, UniRec format (described in subsection 1.2.4) is efficient when storing and transmitting data but inconvenient for reporting purposes since it is an unreadable binary protocol. CESNET uses Warden⁶ — system for sharing information about detected events and threats, and Mentat⁷ as the web interface for easy access and visualization of information from Warden. The proposed architecture for deploying the miner detector module is shown on the figure 3.12. *Miner aggregator* is a module for aggregating raw outputs of the miner detector, which generates alerts after certain conditions are met. *IDEA reporter* is a module that uses information from *miner aggregator* alerts to craft messages for the Warden system. Both modules are described in the following subsections.

3.7.1 Miner aggregator

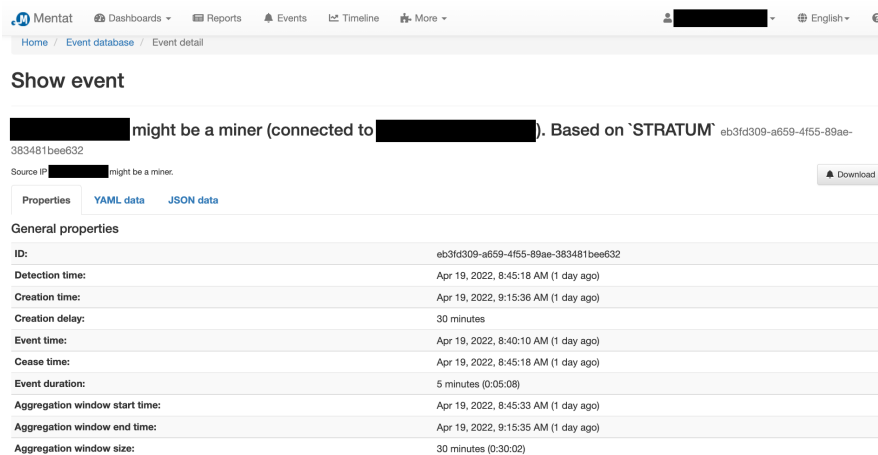
Miner aggregator is a module designed to collect raw outputs of the miner detector, aggregate them and send an alert to its output after certain conditions are met. A flow marked as a miner is received from the miner detector, a flow key is calculated and either a new record is added to the flow cache or the existing one is updated. A flow key is calculated as SHA-256 hash from source and destination IP addresses and ports. Each record in the cache stores the overall counts of transmitted packets, bytes and also reasons why flows were marked as the miners. Possible values are *STRATUM*, when Stratum was matched, *DST*, if TLS SNI was present and the result was obtained by DST combination of TLS SNI score and ML probability, or *ML* if only the ML probability was used.

Cache record is exported from the flow cache and sent to the output if one of the two conditions is met. The first condition has a similar purpose as the active timeout in flow exporting. It is meant for exporting records with a large number of flows — after a certain maximum is reached, the record is exported. We set this maximum to 5, mainly for debugging purposes. The second condition helps with records that would not reach the maximum threshold and it is similar to passive timeout. After a certain time of inactivity, record is exported from the cache. Experimentally, we set this timeout to 30 minutes. Both timeouts can be set via the module's arguments. When a record is exported, it is deleted from the cache.

Miner aggregator is implemented again as the NEMEA module, which uses UniRec format on its input and output. When a cache record is exported, the output UniRec message contains total bytes and packets and IP addresses. It also contains the aggregation window (time window during which base flows were collected) and detection reason which dominated.

⁶www.warden.cesnet.cz/en/about_project

⁷www.mentat.cesnet.cz



383481bee632 might be a miner (connected to [redacted]), Based on 'STRATUM' eb3fd309-a659-4f55-89ae-383481bee632

Source IP [redacted] might be a miner. [Download](#)

Properties [YAML data](#) [JSON data](#)

General properties

ID:	eb3fd309-a659-4f55-89ae-383481bee632
Detection time:	Apr 19, 2022, 8:45:18 AM (1 day ago)
Creation time:	Apr 19, 2022, 9:15:36 AM (1 day ago)
Creation delay:	30 minutes
Event time:	Apr 19, 2022, 8:40:10 AM (1 day ago)
Cease time:	Apr 19, 2022, 8:45:18 AM (1 day ago)
Event duration:	5 minutes (0:05:08)
Aggregation window start time:	Apr 19, 2022, 8:45:33 AM (1 day ago)
Aggregation window end time:	Apr 19, 2022, 9:15:35 AM (1 day ago)
Aggregation window size:	30 minutes (0:30:02)

■ **Figure 3.13** Screenshot of IDEA alert with a potential miner in the Mentat web interface

3.7.2 IDEA reporter

IDEA reporter is a very simple module, which was created from the existing template⁸. After an alert from miner aggregator is received, a message is crafted by filling data into the predefined JSON structure (called IDEA format⁹). The message is then passed on to the Warden system and is available in the Mentat web interface for manual inspection by the network administrators. IDEA message contains IP addresses, source detector, type of detection and more. It also provides communication statistics, such as bytes, packets and aggregation windows which are described in the subsection 3.7.1. An alert in the Mentat web interface created by our setup is shown on the figure 3.13.

⁸www.github.com/CESNET/Nemea-Modules/blob/master/report2idea/template.py

⁹www.idea.cesnet.cz/en/index

Evaluation

This chapter describes the back checking of how our datasets are labeled based on the implemented detector. The quality of created datasets is evaluated. Furthermore, several performance metrics used for the final evaluation are also described. Details from the deployment of our detector and its architecture on the CESNET2 network are also provided.

4.1 Datasets' quality

Since everyone can create their own cryptocurrency or a mining pool server, it is impossible to create a blacklist that would contain every single one of them. Therefore, there is a very high possibility that our dataset representing the non-miner class contains flows generated by miners. To address this issue, we used our support classifiers — Stratum detector and TLS SNI classifier — to look for either flows containing Stratum or flows where TLS SNI contained suspicious keywords and after manual review, we fixed several labels. But even after this there is still a chance of mislabels in our datasets.

In the first step, we generated predictions with the Stratum detector and then filtered only non-miner flows. Then, we manually reviewed the results, but unfortunately there were no flows with these characteristics. In the second step, we used the TLS SNI classifier to find flows with suspicious hosts in the TLS SNI field. This produced several hundred flows that turned out to be mislabeled. We fixed labels of around 800 flows and marked them as miners. However, we still cannot be sure if all non-miner flows have true labels.

4.1.1 Permutation tests

Permutation tests are used for the evaluation of datasets' quality [82]. As described by Camacho et al. [82], permutation tests can be performed in various ways but we will mainly focus on the permutation of labels. This is because label permutation is more flexible and also other variants of permutation tests would require larger computational effort [82]. By random permutations of labels we can test if there is any correlation between the labels and features in a dataset.

As also described in [82], we firstly train several ML models and obtain a pool of performance measures for the tested dataset. Then, we apply label permutation to percentages of data (ranges from 1% to 100%). After that, we measure the loss of ML models' performance. Finally, P -values are calculated by the formula shown on 4.1, where $F1$ is the $F1$ -Score of real data and $F1^*$ is $F1$ -Score of permuted data. Other performance metrics may be used as well. Low P -values

show a correlation between features and labels and therefore infer the good quality of data in the dataset.

$$P = (\text{No. of } (F1^* \geq F1) + 1) / (\text{Total No. of } F1^* + 1) \quad (4.1)$$

Because even permutation tests based on the label permutation are demanding for time and computational resources, we decided to only run these tests on samples of our datasets. We used following ML models for obtaining the performance pool: k -nearest neighbors (KNN), Support Vector Machine (SVN), Decision tree (DT), Random forest (RF), AdaBoost (AB), XGBoost (XGB) and Multi-layer perceptron (MLP).

We performed permutation tests of labels under the supervision of Ing. Dominik Soukup. Used samples contained randomly selected 5000 flows per class, totalling 10 000 flows per dataset. The number of permutations was set to 200 and permutations were applied to 50%, 25%, 10%, 5% and 1% of the data. Performance drops of all ML models were around 30% for all datasets. AdaBoost model scored P -value of 0.1542 during the 1% permutation of labels from dataset 04. We suspect that this might be caused by the inept permutation of a small amount of labels. However, all other P -values were below 0.05. In conclusion, permutation tests showed a good overall quality of all datasets. See appendix B for performance drops figures and tables with P -values.

4.2 Performance metrics

One of the basic principles when evaluating the ML models is a group of true and false positives and negatives. As defined in [83]:

1. True positive (TP): model correctly predicts positive class
2. False positive (FP): model incorrectly predicts positive class
3. False negative (FN): model incorrectly predicts negative class
4. True negative (TN): model correctly predicts negative class

Many other evaluation techniques use true/false positives/negatives. Raschka [84] defined Confusion matrix for binary prediction as a 2×2 matrix with displaying numbers of “actual” and “predicted”. Shown numbers can be both absolute or relative. Accuracy (ACC) metric shows the number of correct predictions [84]. This metric is calculated as the sum of correct predictions divided by the total number of predictions.

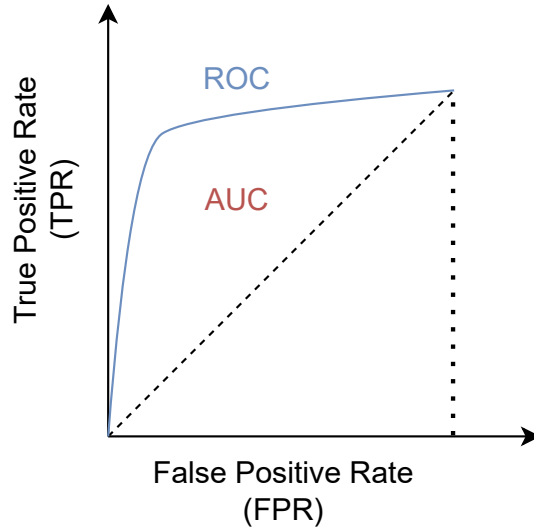
$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

As also described by Raschka [84], Precision (PRE) and Recall (REC) are metrics that are more commonly used and are based on the true and false-positive rates. Recall is also called Sensitivity and Precision is called Specificity. The $F1$ -Score combines both Precision and Recall.

$$PRE = \frac{TP}{TP + FP}$$

$$REC = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$



■ **Figure 4.1** ROC AUC metrics example

■ **Table 4.1** Performance of the chosen Random forest model

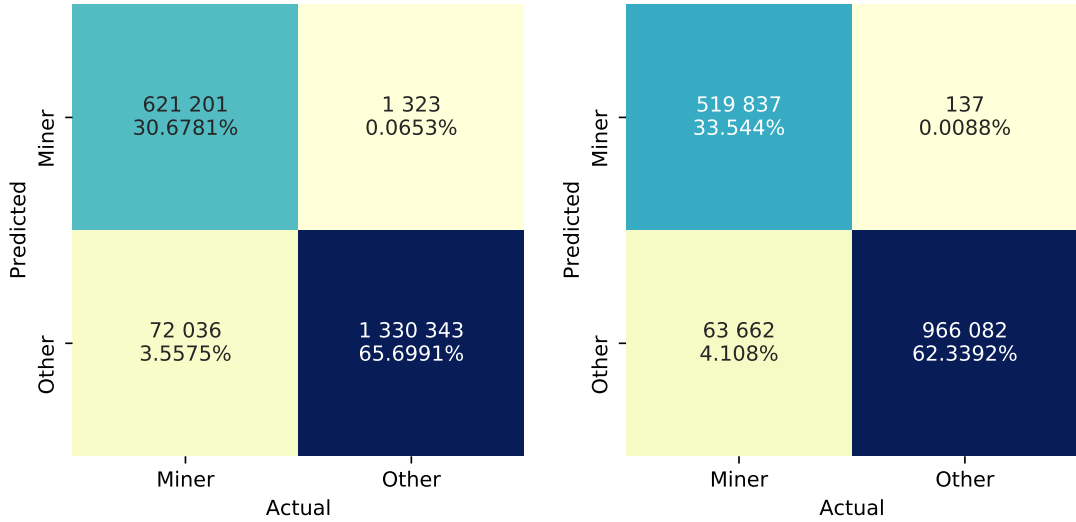
Dataset	Purpose	ROC AUC	Accuracy	Precision	Recall	F1-Score
01	Verification	99.11%	99.29%	99.61%	98.44%	99.02%
02	Train, Test	99.14%	99.27%	99.08%	98.74%	98.91%
03	Verification	97.47%	98.12%	98.07%	95.78%	96.91%
04	Verification	97.37%	97.77%	98.22%	95.77%	96.98%

Receiver Operator Characteristic (ROC) graphs, defined in [84], are used for visualization of models' performance. Models with performance below ROC's diagonal are considered to be worse than *random guessing*. A perfect classifier has no false predictions and would be located in the top left corner of the ROC graph. Area Under the (ROC) curve, called *AUC*, can be calculated and used to express the model's performance. Example is shown on the figure 4.1.

4.3 Performance of the miner detector

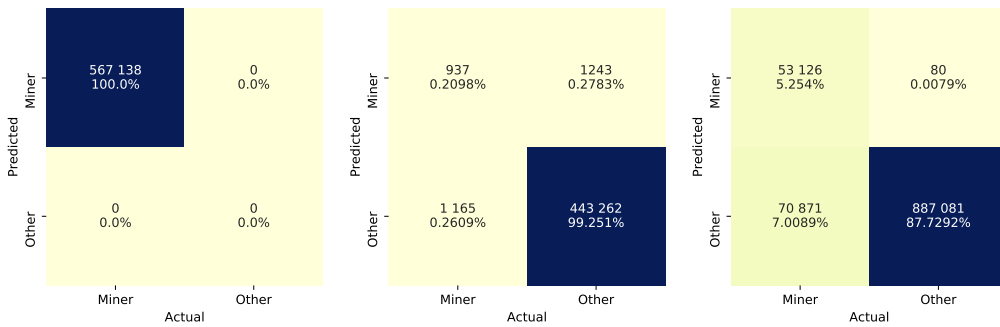
When evaluating ML models' performance, we used several metrics described in section 4.2. Random forest turned out to be the best option and achieved *F1-Score* more than 96.5% on each dataset. It was trained on dataset 02, data captured in the first half of January, 2022. The evaluation was done on all data from each dataset and ROC AUC, *F1-Score*, Recall, Precision and Accuracy were used, shown in the table 4.1. Hyperparameters of other models' configurations and their performance are shown in the appendix C. When selecting the best model, we considered all metrics, but *F1-Score* was considered the most important one.

To evaluate implemented prototype of the miner detector, we merged all the datasets together to get one finalized dataset. The detector was then used to detect miner flows in this finalized dataset and metrics were calculated. The evaluation was performed with all flows from the finalized datasets, the confusion matrix is shown on the figure 4.2a. During the second evaluation, the detector only processed flows that have more than seven packets in each direction, shown on the figure 4.2b. Our detector achieved an accuracy of 96.3772% when all flows were analyzed and **95.882%** when the pre-filter was in place. It may seem that the pre-filter worsened the



(a) Confusion matrix when all data were used (b) Confusion matrix when the pre-filter was in place

■ **Figure 4.2** Confusion matrices of miner detector (the Meta classifier)



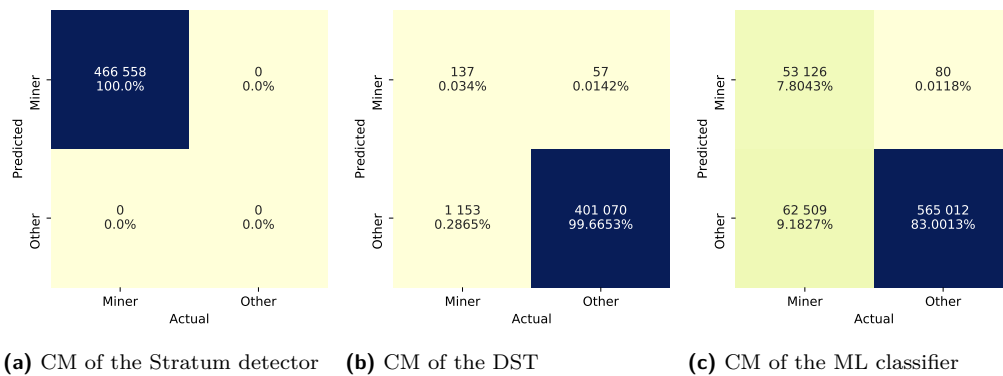
(a) CM of the Stratum detector (b) CM of the DST (c) CM of the ML classifier

■ **Figure 4.3** Confusion matrices of support classifiers when all data were used

accuracy of our detector, however it significantly lowered the number of false positives. This is a very desirable effect for the high-speed networks because large number of false-positive alerts would have to be manually reviewed by network administrators.

Moreover, we tracked how many flows were processed by support classifiers and their results. Statistics of the Stratum detector, DST combination of the TLS SNI classifier and the ML classifier, and only by the ML classifier are shown on the figure 4.3 and figure 4.4.

Lastly, we evaluated the speed of the detector. The detector achieved total speed of around 30 000 flows per second when run on our laptop in WSL2 with Intel i5-7200, 2.5 GHz. Default buffer size of 100 000 flows was used. However, it was able to process **41 500** flows per second when tested on AMD Ryzen 3700X, 3.59 GHz.



■ **Figure 4.4** Confusion matrices of support classifiers when the pre-filter was in place

4.4 Deployment on CESNET

Three implemented modules — miner detector, miner aggregator and reporter were deployed on the national CESNET network on 01.04.2022. Since that time, almost 100 000 alerts have been generated and only one false positive has been found so far. To our surprise, most of the alerts were based on Stratum detection, but around 300 alerts were based on the DST (TLS SNI and ML). However, a different version of the Scikit-learn library was installed on the server where the detector runs. This could potentially cause an incorrect load of the ML module from the file and cause the ML classifier to work incorrectly. We installed the correct version of the library, but it did not help. Initially, the ML threshold was set to 1.0, but we later discovered that this value is very strict and caused that no flows were marked as miners by this classifier. ML threshold was therefore set to 0.997 and eventually the ML classifier started producing results as well. The detector is now deployed on the national CESNET2 network and is successfully detecting miner flows. Without a doubt, a lot of miner flows remain undetected but at the same time network administrators are not drowned in the false alerts.

Conclusion

The main goal of this thesis was to design an algorithm for the automatic detection of cryptominers. Since cryptomining may be performed by cybercriminals for their own enrichment in an abusive way, it is essential to be able to detect it. The main goal also included developing a software prototype capable of processing real high-speed network traffic using the NEMEA system. Previous attempts were able to detect miner communication, however this thesis explored real high-speed networks, which has not been studied very thoroughly in the past.

We created the collection of datasets containing miner and non-miner traffic and also a tool that is able to adapt to the changing nature of mining pools structure and automatically capture miner traffic over time. We analyzed this traffic and proposed three possible ways of miner detection, based on the Stratum mining protocol for unencrypted traffic, suspicious hostnames in the TLS SNI and ML for encrypted traffic. These three detectors were used together to create a highly accurate ensemble detector.

Moreover, we implemented our ensemble detector in Python with an emphasis on efficiency. At this point, our implementation is deployed in the CESNET infrastructure protecting nation network CESNET2 with half a million users. It also protects the national computational grid MetaCentrum against abuse. More than 100 000 alerts were generated with a minimum of false positives.

In the future, we plan to make the developed detector more customizable for users, let them set the keywords to match in the TLS SNI and also to automatically re-train the ML model used by the ML classifier. It is expected that Stratum V2 will become a new standard in mining protocols. Since we mainly focused on Stratum V1, a possible extension of our work is to focus on Stratum V2 and add another support classifier for this protocol.

Selected mining pools

This appendix provides overview of selected mining pools and their extracted mining protocols and ports used for mining. Table A.1 holds information about BTC mining pools, table A.2 for ETH and table A.3 for XMR. Value “Stratum” in the column *Protocol* means that the Stratum was directly specified as the used mining protocol. Value “Stratum?” means that we found Stratum in the “Help” section or sections with example configurations of mining software.

■ **Table A.1** Selected mining pools for BTC

No.	Pool	Protocol	Ports
1.	www.f2pool.com	Stratum	25, 3333, 1314
2.	www.v3.antpool.com	Stratum	25, 443, 3333
3.	www.viabtc.com	Stratum	25, 443, 3333
4.	www.poolin.com	Stratum	443, 700, 1883
5.	www.foundrydigital.com	—	—
6.	www.pool.btc.com	Stratum	25, 443, 1800
7.	www.pool.binance.com	Stratum	443, 1800, 3333, 8888
8.	www.slushpool.com	Stratum	3333
9.	www.hpt.com	—	—
10.	www.marathondh.com	—	—

■ **Table A.2** Selected mining pools for ETH

No.	Pool	Protocol	Ports
1.	www.ethermine.org	Stratum	4444, 5555, 14444
2.	www.f2pool.com	Stratum	6688
3.	www.hiveon.net	Stratum	4444, 14444, 24443
4.	www.eth.nanopool.org	Stratum, Getwork	8888, 9433, 9999
5.	www.beepool.com/coindetail/eth	Stratum	9630, 9531, 9532
6.	www.eth.2miners.com	Stratum?	2020, 12020
7.	www.flexpool.io	Stratum	4444, 5555, 14444
8.	www.pool.binance.com	Stratum	25, 443, 1800, 3333, 8888
9.	www.ethereum.miningpoolhub.com	Stratum	20535
10.	www.v3.antpool.com	Stratum	25, 443, 8008

■ **Table A.3** Selected mining pools for XMR

No.	Pool	Protocol	Ports
1.	www.minexmr.com	Stratum?	443, 3333, 4444
2.	www.supportxmr.com	Stratum?	80, 443, 8080, 3333, 5555, 7777, 9999
3.	www.xmr.nanopool.org	Stratum	14443, 14444
4.	www.f2pool.com	Stratum	13531
5.	www.c3pool.com	Stratum	80, 443, 13333, 15555, 17777, 23333
6.	www.web.xmrpool.eu	Stratum	443, 3333, 5555, 7777, 9999
7.	www.hashcity.org	Stratum	2321, 3100, 4444, 4445
8.	www.xmr.2miners.com	Stratum	2222, 3333, 12222, 13333
9.	www.p2pool.io	Stratum	18080, 37889
10.	www.monero.hashvault.pro	Stratum	80, 443, 3333, 5555, 7777, 8888

Permutation test results

This appendix provides detailed results of permutation tests run on our datasets 01-04. Tables B.1, B.2, B.3 and B.4 show resulting P -values for each dataset. Moreover, figure B.1 shows how ML models performance dropped after the label permutations.

■ **Table B.1** P -values for dataset 01 from permutation tests with 200 permutations

Model	50%	25%	10%	5%	1%
KNN	.0050	.0050	.0050	.0050	.0050
SVM	.0050	.0050	.0050	.0050	.0050
DT	.0050	.0050	.0050	.0050	.0050
RF	.0050	.0050	.0050	.0050	.0050
AB	.0050	.0050	.0050	.0050	.0050
XGB	.0050	.0050	.0050	.0050	.0050
MLP	.0050	.0050	.0050	.0050	.0050

■ **Table B.2** P -values for dataset 02 from permutation tests with 200 permutations

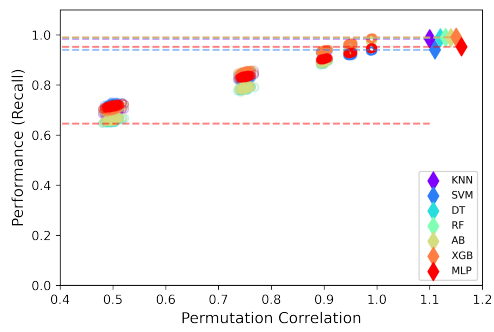
Model	50%	25%	10%	5%	1%
KNN	.0050	.0050	.0050	.0050	.0050
SVM	.0050	.0050	.0050	.0050	.0050
DT	.0050	.0050	.0050	.0050	.0050
RF	.0050	.0050	.0050	.0050	.0050
AB	.0050	.0050	.0050	.0050	.0050
XGB	.0050	.0050	.0050	.0050	.0050
MLP	.0050	.0050	.0050	.0050	.0398

■ **Table B.3** P -values for dataset 03 from permutation tests with 200 permutations

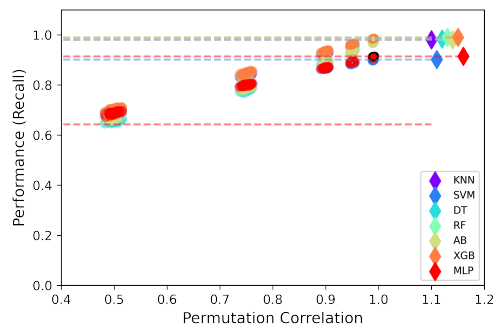
Model	50%	25%	10%	5%	1%
KNN	.0050	.0050	.0050	.0050	.0050
SVM	.0050	.0050	.0050	.0050	.0050
DT	.0050	.0050	.0050	.0050	.0050
RF	.0050	.0050	.0050	.0050	.0050
AB	.0050	.0050	.0050	.0050	.0149
XGB	.0050	.0050	.0050	.0050	.0050
MLP	.0050	.0050	.0050	.0050	.0050

■ **Table B.4** P -values for dataset 04 from permutation tests with 200 permutations

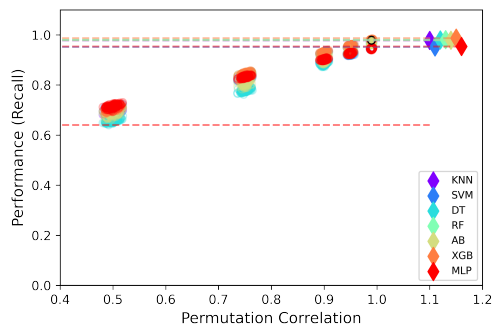
Model	50%	25%	10%	5%	1%
KNN	.0050	.0050	.0050	.0050	.0050
SVM	.0050	.0050	.0050	.0050	.0050
DT	.0050	.0050	.0050	.0050	.0050
RF	.0050	.0050	.0050	.0050	.0050
AB	.0050	.0050	.0050	.0050	.1542
XGB	.0050	.0050	.0050	.0050	.0050
MLP	.0050	.0050	.0050	.0050	.0249



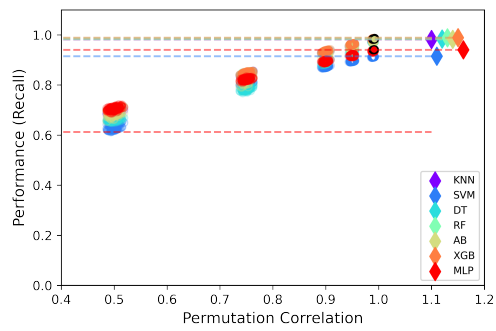
(a) Performance drops on the dataset 01



(b) Performance drops on the dataset 02



(c) Performance drops on the dataset 03



(d) Performance drops on the dataset 04

■ **Figure B.1** Performance drops of ML models used for permutation testing

Appendix C

ML models' results

This appendix provides detailed results of remaining ML models tested when looking for a suitable model. Tables C.1 and C.3 show hyperparameters of Decision tree and Random forest models, column *Name* is then used as a key to the tables C.2 and C.4, which show different performance metrics for each model and dataset. Symbol “*” in the column *Dataset* means that corresponding model was trained on this dataset, in tables C.2 and C.4.

■ **Table C.1** Overview of Decision tree models

Name	Criterion	Max depth	Max feat.	Min leaf	Min split	Splitter
DT1	entropy	7	None	0.005	0.01	best
DT2	entropy	6	None	0.005	0.005	best
DT3	entropy	8	\log_2	0.1	0.15	best
DT4	entropy	6	auto	0.1	0.15	best
DT5	entropy	20	\log_2	0.1	0.2	best
DT6	gini	6	auto	0.1	0.25	best
DT7	gini	20	None	2	7	best
DT8	entropy	None	None	2	5	best

■ **Table C.2** Overview of Decision tree models' performance

Name	Dataset	ROC AUC	Accuracy	Precision	Recall	F1-Score
DT1	01*	97.26%	97.65%	97.63%	95.84%	96.73%
DT1	02	90.98%	93.09%	94.40%	84.51%	89.18%
DT1	03	92.06%	93.27%	89.18%	88.91%	89.04%
DT1	04	90.28%	91.90%	93.82%	83.87%	88.57%
DT2	01	96.89%	97.35%	97.46%	95.19%	96.31%
DT2	02*	95.78%	96.46%	95.75%	93.66%	94.70%
DT2	03	94.70%	95.72%	93.89%	92.06%	92.97%
DT2	04	94.36%	94.95%	94.37%	91.99%	93.16%
DT3	01*	90.11%	89.80%	82.50%	91.24%	86.65%
DT3	02	85.04%	85.75%	76.71%	82.88%	79.67%
DT3	03	84.47%	83.83%	68.99%	86.15%	76.62%
DT3	04	82.28%	82.51%	74.33%	81.36%	77.68%
DT4	01	91.35%	93.09%	95.40%	85.03%	89.92%
DT4	02*	88.25%	91.06%	92.81%	79.63%	85.72%
DT4	03	91.84%	93.50%	91.01%	87.52%	89.23%
DT4	04	89.58%	91.30%	93.25%	82.74%	87.68%
DT5	01*	88.76%	91.47%	97.01%	78.90%	87.02%
DT5	02	84.63%	88.85%	93.75%	71.70%	81.25%
DT5	03	90.72%	92.98%	91.71%	84.84%	88.14%
DT5	04	88.45%	90.68%	94.68%	79.57%	86.47%
DT6	01	91.72%	93.57%	96.93%	84.97%	90.56%
DT6	02*	87.49%	90.73%	93.90%	77.54%	84.94%
DT6	03	87.14%	88.98%	81.92%	82.35%	82.14%
DT6	04	87.24%	89.48%	92.40%	78.33%	84.79%
DT7	01*	99.75%	99.78%	99.71%	99.67%	99.69%
DT7	02	93.98%	95.76%	98.76%	88.52%	93.36%
DT7	03	98.13%	98.43%	97.52%	97.36%	97.44%
DT7	04	97.81%	98.08%	98.12%	96.73%	97.42%
DT8	01	99.52%	99.60%	99.71%	99.20%	99.45%
DT8	02*	99.47%	99.54%	99.36%	99.26%	99.31%
DT8	03	97.36%	98.12%	98.49%	95.36%	96.90%
DT8	04	94.67%	95.71%	97.84%	90.52%	94.04%

■ **Table C.3** Overview of Random forest models

Name	Criterion	Max depth	Max feat.	Min leaf	Min split	Estm.
RF1	entropy	10	<i>sqrt</i>	2	5	50
RF2	gini	10	<i>sqrt</i>	2	5	100
RF3	gini	6	None	10	10	50
RF4	gini	6	\log_2	10	10	50
RF5	gini	5	None	10	10	50
RF6	gini	5	<i>sqrt</i>	10	10	100

■ **Table C.4** Overview of Random forest models' performance

Name	Dataset	ROC AUC	Accuracy	Precision	Recall	F1-Score
RF1	01*	99.68%	99.73%	99.79%	99.47%	99.63%
RF1	02	93.83%	95.68%	98.91%	88.15%	93.22%
RF1	03	96.45%	97.53%	98.22%	93.66%	95.88%
RF1	04	95.14%	95.53%	99.32%	88.66%	93.69%
RF2	01	99.11%	99.29%	99.61%	98.44%	99.02%
RF2	02*	99.14%	99.27%	99.08%	98.74%	98.91%
RF2	03	94.47%	98.12%	98.07%	95.78%	96.91%
RF2	04	97.37%	97.77%	98.22%	95.77%	96.98%
RF3	01*	98.67%	98.86%	98.91%	97.95%	98.42%
RF3	02	92.53%	94.47%	96.69%	86.56%	91.35%
RF3	03	94.06%	95.52%	94.93%	90.26%	92.54%
RF3	04	91.58%	93.18%	96.11%	85.23%	90.34%
RF4	01	97.83%	98.32%	99.27%	96.06%	97.64%
RF4	02*	97.02%	97.71%	98.21%	94.93%	96.54%
RF4	03	94.03%	95.84%	96.91%	89.32%	92.96%
RF4	04	92.01%	93.84%	98.56%	84.77%	91.14%
RF5	01*	97.98%	98.26%	98.20%	96.98%	97.58%
RF5	02	92.12%	94.07%	95.86%	86.13%	90.74%
RF5	03	93.47%	95.08%	94.40%	89.28%	91.77%
RF5	04	91.26%	92.98%	96.35%	84.44%	90.00%
RF6	01	97.51%	98.02%	98.85%	95.66%	97.22%
RF6	02*	96.40%	97.17%	94.44%	94.06%	95.72%
RF6	03	93.41%	95.15%	95.02%	88.89%	91.86%
RF6	04	91.19%	92.94%	96.41%	84.26%	89.93%

Experimental credibility values

This appendix shows experimental values of Stratum credibility (table D.1) and TLS SNI credibility (table D.2) in the original design of the Stratum detector and the TLS SNI classifier. As described in chapter 3, credibility values were later removed.

■ **Table D.1** Experimental credibility values for Stratum classifier

Client Payload Printable	Server Payload Printable	Credibility
Yes	Yes	0.9
Yes	No	0.6
No	Yes	0.6
No	No	0.0

■ **Table D.2** Experimental credibility values for TLS SNI classifier

Match	Credibility
Full	0.9
Partial	0.6
None	0.0

Appendix E

User manual

This chapter provides a quick overview of the usage of the two implemented modules — detector and aggregator. The IDEA reporter module was created from the template. Therefore, we do not provide a help section in this appendix since needed information is publicly available¹.

Miner Detector

=====

usage:

```
./minerdetector.py [-h] [-m MODEL] [-b BUFFER] [-i I] [-d DST_THRESHOLD]
                  [-t ML_THRESHOLD] [-v]
```

optional arguments:

```
-h, --help
    Show this help message and exit
-m MODEL, --model MODEL
    Pickle file with ML model
-b BUFFER, --buffer BUFFER
    Flow buffer size
-i I
    IFC interfaces for pytrap
-d DST_THRESHOLD, --dst-threshold DST_THRESHOLD
    Threshold for miners' DST pignistic function [0..1]
-t ML_THRESHOLD, --ml-threshold ML_THRESHOLD
    Threshold for ML proba [0..1]
-v, --verify-mode
    Run detector in verification mode, flow labels are required
```

¹www://github.com/CESNET/Nemea-Framework/blob/master/pycommon/report2idea.py

Miner Aggregator

=====

usage:

```
usage: mineraggregator.py [-h] [-e EXPORT_INTERVAL] [-a ACTIVE_TIMEOUT]
                          [-p PASSIVE_TIMEOUT] [-i I]
```

optional arguments:

```
-h, --help
    Show this help message and exit
-e EXPORT_INTERVAL, --export-interval EXPORT_INTERVAL
    Number of seconds the export thread periodically sleeps
-a ACTIVE_TIMEOUT, --active-timeout ACTIVE_TIMEOUT
    Max number of flows, when this number is reached, data are sent
    to out IFC
-p PASSIVE_TIMEOUT, --passive-timeout PASSIVE_TIMEOUT
    Number of minutes, when this number of minutes passed from last
    activity, data are sent to out IFC
-i I
    IFC interfaces for pytrap
```


Bibliography

1. BACIU, Paula. *Czech Prime Minister Accuses Pirate Party of Mining Bitcoin* [online]. 2018 [visited on 2022-04-06]. Available from: <https://bitcoinist.com/prime-minister-accuses-czech-pirate-party-of-mining-bitcoin-so-what/>.
2. MALWAREBYTES. *Malvertising definition* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://www.malwarebytes.com/malvertising>.
3. CIMPANU, Catalin. *Malvertising Campaign Mines Cryptocurrency Right in Your Browser* [online]. 2017 [visited on 2022-04-06]. Available from: <https://www.malwarebytes.com/malvertising>.
4. HRUSKA, Joel. *Browser-Based Mining Malware Found on Pirate Bay, Other Sites* [online]. 2017 [visited on 2022-04-06]. Available from: <https://www.extremetech.com/internet/255971-browser-based-cryptocurrency-malware-appears-online-pirate-bay>.
5. VUIJSJE, Eliana. *Cryptocurrency Malvertising Campaign Hijacks Users' Browsers* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://www.geoedge.com/cryptocurrency-malvertising-campaign-hijacks-users-browsers/>.
6. NAKAMOTO, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System* [online]. [N.d.] [visited on 2021-11-18]. Available from: <https://bitcoin.org/bitcoin.pdf>.
7. PHIPPS, Marian. *How Much of The World's Money Is in Cryptocurrency* [online]. 2022 [visited on 2022-04-01]. Available from: <https://revenuesandprofits.com/money-in-cryptocurrency/>.
8. KOTESKA, Bojana; KARAFILOSKI, Elena; MISHEV, Anastas. *Blockchain Implementation Quality Challenges: A Literature Review*. In: 2017.
9. CROSBY, Michael; PATTANAYAK, Pradan; VERMA, Sanjeev; KALYANARAMAN, Vignesh, et al. *Blockchain technology: Beyond bitcoin*. *Applied Innovation*. 2016, vol. 2, no. 6-10, p. 71.
10. HAYWARD, Andrew. *What Are Privacy Coins? Monero, Zcash, and Dash Explained* [online]. 2021 [visited on 2022-03-14]. Available from: <https://decrypt.co/resources/what-are-privacy-coins-monero-zcash-and-dash-explained>.
11. AVAN-NOMAYO, Osato. *South Korea to Ban Privacy Coins from Q1 2021* [online]. 2020 [visited on 2022-04-04]. Available from: <https://beincrypto.com/south-korea-to-ban-privacy-coins-from-q1-2021/>.
12. GHIMIRE, Suman; SELVARAJ, Henry. *A Survey on Bitcoin Cryptocurrency and its Mining*. In: *2018 26th International Conference on Systems Engineering (ICSEng)*. 2018, pp. 1–6. Available from DOI: 10.1109/ICSENG.2018.8638208.

13. TARMAN, Marko. *What is solo mining and how does it work?* [Online]. 2022 [visited on 2022-04-04]. Available from: <https://www.nicehash.com/blog/post/what-is-solo-mining-and-how-it-works>.
14. BITCOIN PROJECT. *Bitcoin Developer* [online]. [N.d.] [visited on 2022-04-04]. Available from: <https://developer.bitcoin.org/devguide/mining.html>.
15. HERTIG, Alyssa; LEECH, Ollie. *What Does Hashrate Mean and Why Does It Matter?* [Online]. 2021 [visited on 2022-04-04]. Available from: <https://www.coindesk.com/tech/2021/02/05/what-does-hashrate-mean-and-why-does-it-matter/>.
16. TU8RNER, Brian; DEMURO, Jonas P. *Best mining software of 2022* [online]. 2021 [visited on 2022-04-05]. Available from: <https://www.techradar.com/best/mining-software>.
17. SLUSHPOOL. *Stratum V1* [online] [visited on 2021-11-18]. Available from: <https://brains.com/stratum-v1>.
18. SLUSHPOOL. *Stratum V2 mining URLs and guide* [online] [visited on 2021-11-18]. Available from: <https://help.slushpool.com/en/support/solutions/articles/77000423566-stratum-v2-mining-urls-and-guide>.
19. POOLIN.COM. *TLS protocol applied in Cryptocurrency mining* [online]. 2021 [visited on 2022-04-05]. Available from: <https://medium.com/poolin/tls-protocol-applied-in-cryptocurrency-mining-9de15c08c405>.
20. CERF, V.; KAHN, R. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*. 1974, vol. 22, no. 5, pp. 637–648. Available from DOI: 10.1109/TCOM.1974.1092259.
21. POSTEL, Jon et al. *Transmission Control Protocol* [RFC 793]. RFC Editor, 1981. Request for Comments, no. 793. Available from DOI: 10.17487/RFC0793.
22. LEUNG, Ka-cheong; LI, Victor O.k.; YANG, Daiqin. An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges. *IEEE Transactions on Parallel and Distributed Systems*. 2007, vol. 18, no. 4, pp. 522–535. Available from DOI: 10.1109/TPDS.2007.1011.
23. DIERKS, Tim; RESCORLA, Eric. The transport layer security (TLS) protocol version 1.2. 2008.
24. OGATA, K.; FUTATSUGI, K. Equational Approach to Formal Analysis of TLS. In: *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. 2005, pp. 795–804. Available from DOI: 10.1109/ICDCS.2005.32.
25. MOCKAPETRIS, Paul et al. Domain names-implementation and specification. 1987.
26. CLOUDFLARE. *DNS over TLS vs. DNS over HTTPS* [online]. [N.d.] [visited on 2022-04-05]. Available from: <https://www.cloudflare.com/en-gb/learning/dns/dns-over-tls/>.
27. VESELÝ, Vladimír; ŽÁDNÍK, Martin. How to detect cryptocurrency miners? By traffic forensics! *Digital Investigation*. 2019, vol. 31, p. 100884. ISSN 1742-2876. Available from DOI: <https://doi.org/10.1016/j.diin.2019.08.002>.
28. STRATUM PLATFORM. Stratum — network protocol specification: draft. In: [online]. 2011 [visited on 2022-03-13]. Available from: https://docs.google.com/document/d/17zHy1SULhgtCMbyp08cHgpWH73V5iUQKk_OrWvMqSNs/edit.
29. PASTRANA, Sergio; SUAREZ-TANGIL, Guillermo. A First Look at the Crypto-Mining Malware Ecosystem: A Decade of Unrestricted Wealth. In: *Proceedings of the Internet Measurement Conference*. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 73–86. IMC '19. ISBN 9781450369480. Available from DOI: 10.1145/3355369.3355576.

30. KHATRI, Yogita. *Crypto Mining Malware Has Netted Nearly 5% of All Monero, Says Research* [online]. 2019 [visited on 2022-03-14]. Available from: <https://www.coindesk.com/markets/2019/01/10/crypto-mining-malware-has-netted-nearly-5-of-all-monero-says-research/>.
31. MCAFFEE. *McAfee Labs Threats Report: December 2018* [online]. 2018 [visited on 2022-03-14]. Available from: <https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf>.
32. NELSON, Jason. *This Monero Malware Is Targeting Enterprise Networks* [online]. 2021 [visited on 2022-03-14]. Available from: <https://decrypt.co/87485/this-monero-malware-is-targeting-enterprise-networks>.
33. GALLAGHER, Sean. *Two flavors of Tor2Mine miner dig deep into networks with PowerShell, VBScript* [online]. 2021 [visited on 2022-03-14]. Available from: <https://news.sophos.com/en-us/2021/12/02/two-flavors-of-tor2mine-miner-dig-deep-into-networks-with-powershell-vbscript/>.
34. SPEZZA, Gianluca. *North Korea and the Role of Science Diplomacy* [online]. 2022 [visited on 2022-03-14]. Available from: <https://isdp.eu/content/uploads/2022/01/North-Korea-and-the-Role-of-Science-Diplomacy-24.01.2022.pdf>.
35. LIU, Jingqiang; ZHAO, Zihao; CUI, Xiang; WANG, Zhi; LIU, Qixu. A Novel Approach for Detecting Browser-Based Silent Miner. In: *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*. 2018, pp. 490–497. Available from DOI: 10.1109/DSC.2018.00079.
36. LIU, Zhaoyan; CHEN, Binfa; LI, Zhen; LI, Hui; WANG, Dengzheng; LYU, Yang; GAO, Lu; HOU, Bingxu. Bitcoin Mining Recognition Based on Community Detection with Electricity Consumption Data. In: *2021 IEEE 5th Conference on Energy Internet and Energy System Integration (EI2)*. 2021, pp. 3091–3096. Available from DOI: 10.1109/EI252483.2021.9713449.
37. SWEDAN, AbedAlqader; KHUFFASH, Ahmad N.; OTHMAN, Othman; AWAD, Ahmed. Detection and Prevention of Malicious Cryptocurrency Mining on Internet-Connected Devices. In: *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*. Amman, Jordan: Association for Computing Machinery, 2018. ICFNDS '18. ISBN 9781450364287. Available from DOI: 10.1145/3231053.3231076.
38. KHARRAZ, Amin; MA, Zane; MURLEY, Paul; LEVER, Charles; MASON, Joshua; MILLER, Andrew; BORISOV, Nikita; ANTONAKAKIS, Manos; BAILEY, Michael. Outguard: Detecting In-Browser Covert Cryptocurrency Mining in the Wild. In: *The World Wide Web Conference*. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 840–852. WWW '19. ISBN 9781450366748. Available from DOI: 10.1145/3308558.3313665.
39. MUÑOZ, Jordi Zayuelas i; SUÁREZ-VARELA, José; BARLET-ROS, Pere. Detecting cryptocurrency miners with NetFlow/IPFIX network measurements. In: *2019 IEEE International Symposium on Measurements Networking (MN)*. 2019, pp. 1–6. Available from DOI: 10.1109/IWMN.2019.8804995.
40. MANTISNET. *Understanding the Difference Between Network Monitoring and Network Security Monitoring* [online]. [N.d.] [visited on 2022-04-05]. Available from: <https://www.mantisnet.com/blog/understanding-the-difference-between-network-monitoring-and-network-security-monitoring>.
41. GRUBERGER, David. *Network Monitoring vs. Network Security Monitoring* [online]. 2021 [visited on 2022-04-05]. Available from: <https://vulcan.io/blog/network-security-monitoring/>.

42. LEE, Sihyung; LEVANTI, Kyriaki; KIM, Hyong S. Network monitoring: Present and future. *Computer Networks* [online]. 2014, vol. 65, pp. 84–98 [visited on 2022-03-18]. ISSN 1389-1286. Available from DOI: <https://doi.org/10.1016/j.comnet.2014.03.007>.
43. APPNETA. *Active & Passive Network Monitoring: Why enterprises need both* [online]. [N.d.] [visited on 2022-03-19]. Available from: <https://www.appneta.com/pdf/whitepapers/active-passive-network-monitoring-why-enterprises-need-both.pdf>.
44. SVOBODA, Jakub; GHAFIR, Ibrahim; PRENOSIL, Vaclav, et al. Network monitoring approaches: An overview. *Int J Adv Comput Netw Secur*. 2015, vol. 5, no. 2, pp. 88–93.
45. BROOK, Chris. *What is Deep Packet Inspection? How It Works, Use Cases for DPI, and More* [online]. 2018 [visited on 2022-04-05]. Available from: <https://digitalguardian.com/blog/what-deep-packet-inspection-how-it-works-use-cases-dpi-and-more>.
46. HOFFMAN, Chris. *How the “Great Firewall of China” Works to Censor China’s Internet* [online] [visited on 2022-02-21]. Available from: <https://www.howtogeek.com/162092/htg-explains-how-the-great-firewall-of-china-works/>.
47. ASHOOR, Asmaa Shaker; GORE, Sharad. Difference between intrusion detection system (IDS) and intrusion prevention system (IPS). In: *International Conference on Network Security and Applications*. 2011, pp. 497–501.
48. FUCHSBERGER, Andreas. Intrusion detection systems and intrusion prevention systems. *Information Security Technical Report*. 2005, vol. 10, no. 3, pp. 134–139.
49. CROTTI, Manuel; GRINGOLI, Francesco; PELOSATO, Paolo; SALGARELLI, Luca. A statistical approach to IP-level classification of network traffic. In: *2006 IEEE International Conference on Communications*. 2006, vol. 1, pp. 170–176. Available from DOI: 10.1109/ICC.2006.254723.
50. CLAISE, B.; TRAMMELL, B.; AITKEN, P. Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of flow information: Internet Engineering Task Force [online]. 2013 [visited on 2021-11-18]. ISSN 2070-1721. Available from: <https://www.ietf.org/rfc/rfc7011.txt>.
51. HOFSTEDE, Rick; CELEDA, Pavel; TRAMMELL, Brian; DRAGO, Idilio; SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX [online]. 2014, vol. 16, no. 4, pp. 2037–2064 [visited on 2021-11-18]. ISSN 1553-877X. Available from DOI: 10.1109/COMST.2014.2321898.
52. TRAMMELL, Brian; BOSCHI, Elisa. An introduction to IP flow information export (IPFIX). *IEEE Communications Magazine*. 2011, vol. 49, no. 4, pp. 89–95. Available from DOI: 10.1109/MCOM.2011.5741152.
53. CEJKA, Tomas; BARTOS, Vaclav; SVEPES, Marek; ROSA, Zdenek; KUBATOVA, Hana. NEMEA: A framework for network traffic analysis. In: *2016 12th International Conference on Network and Service Management (CNSM)*. 2016, pp. 195–201. Available from DOI: 10.1109/CNSM.2016.7818417.
54. MAHESH, Batta. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR)*. [Internet]. 2020, vol. 9, pp. 381–386.
55. GONG, Zhiqiang; ZHONG, Ping; HU, Weidong. Diversity in Machine Learning. *IEEE Access*. 2019, vol. 7, pp. 64323–64350. Available from DOI: 10.1109/ACCESS.2019.2917620.
56. IBM CLOUD EDUCATION. *Machine Learning* [online]. 2020 [visited on 2022-03-28]. Available from: <https://www.ibm.com/cloud/learn/machine-learning>.
57. MINKA, Thomas. Automatic Choice of Dimensionality for PCA. In: *Advances in Neural Information Processing Systems*. MIT Press, 2000, vol. 13. Available also from: <https://proceedings.neurips.cc/paper/2000/file/7503cfacd12053d309b6bed5c89de212-Paper.pdf>.

58. SAGI, Omer; ROKACH, Lior. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. 2018, vol. 8, no. 4, e1249.
59. PYKES, Kurtis. *The Difference Between Classification and Regression in Machine Learning* [online]. 2021 [visited on 2022-04-05]. Available from: <https://towardsdatascience.com/the-difference-between-classification-and-regression-in-machine-learning-4ccdb5b18fd3>.
60. FIX, Evelyn; HODGES, Joseph Lawson. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*. 1989, vol. 57, no. 3, pp. 238–247.
61. TRIGUERO, Isaac; GARCIA-GIL, Diego; MAILLO, Jesus; LUENGO, Julian; GARCIA, Salvador; HERRERA, Francisco. Transforming big data into smart data: An insight on the use of the k-nearest neighbors algorithm to obtain quality data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. 2019, vol. 9, no. 2, e1289.
62. TSANGARATOS, Paraskevas; ILIA, Ioanna. Comparison of a logistic regression and Naïve Bayes classifier in landslide susceptibility assessments: The influence of models complexity and training dataset size. *CATENA*. 2016, vol. 145, pp. 164–179. ISSN 0341-8162. Available from DOI: <https://doi.org/10.1016/j.catena.2016.06.004>.
63. ARYA, Nisha. *Linear vs Logistic Regression: A Succinct Explanation* [online]. 2022 [visited on 2022-04-05]. Available from: <https://www.kdnuggets.com/2022/03/linear-logistic-regression-succinct-explanation.html>.
64. GUPTA, Prashant. *Decision Trees in Machine Learning* [online]. 2017 [visited on 2022-04-05]. Available from: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>.
65. BREIMAN, Leo. Random forests. *Machine learning*. 2001, vol. 45, no. 1, pp. 5–32.
66. AKAR, Özlem; GÜNGÖR, Oguz. Classification of multispectral images using Random Forest algorithm. *Journal of Geodesy and Geoinformation*. 2012, vol. 1, no. 2, pp. 105–112.
67. SCHAPIRE, Robert E. Explaining adaboost. In: *Empirical inference*. Springer, 2013, pp. 37–52.
68. DEMPSTER, A. P. Upper and Lower Probabilities Induced by a Multivalued Mapping. *The Annals of Mathematical Statistics*. 1967, vol. 38, no. 2, pp. 325–339. Available from DOI: [10.1214/aoms/1177698950](https://doi.org/10.1214/aoms/1177698950).
69. SHAFER, Glenn. *A Mathematical Theory of Evidence*. Princeton University Press, 2021. ISBN 9780691214696. Available from DOI: [doi:10.1515/9780691214696](https://doi.org/10.1515/9780691214696).
70. SHAPIRO, Stuart C. The Dempster-Shafer theory. [N.d.]. Available also from: <http://www.glennshafer.com/assets/downloads/articles/article48.pdf>.
71. BEZERRA, E. D. C.; TELES, A. S.; COUTINHO, L. R.; SILVA E SILVA, F. J. da. Dempster-Shafer Theory for Modeling and Treating Uncertainty in IoT Applications Based on Complex Event Processing. *Sensors (Basel)*. 2021, vol. 21, no. 5.
72. SMETS, Philippe. Data fusion in the transferable belief model. In: *Proceedings of the third international conference on information fusion*. 2000, vol. 1, PS21–PS33.
73. SMETS, Philippe; KENNES, Robert. The transferable belief model. *Artificial intelligence*. 1994, vol. 66, no. 2, pp. 191–234.
74. SHAFER, Glenn. *Dempster-Shafer Theory* [online]. [N.d.] [visited on 2022-04-19]. Available from: <http://fitelson.org/topics/shafer.pdf>.
75. YE, Zhouteng. *An Example of Dempster-Shafer Theory* [online]. 2018 [visited on 2022-04-19]. Available from: <https://zyeeee.wordpress.com/2018/06/06/an-example-of-dempster-shafer-theory/>.

76. CHATZIGIANNIS, Panagiotis; BALDIMTSI, Foteini; GRIVA, Igor; LI, Jiasun. Diversification across mining pools: optimal mining strategies under PoW. *Journal of Cybersecurity*. 2022, vol. 8, no. 1. ISSN 2057-2085. Available from DOI: 10.1093/cybsec/tyab027. tyab027.
77. COIN MARKET CAP. *Today's Cryptocurrency Prices by Market Cap* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://coinmarketcap.com/>.
78. COINBASE. *What is market cap?* [Online]. [N.d.] [visited on 2022-04-06]. Available from: <https://www.coinbase.com/learn/crypto-basics/what-is-market-cap>.
79. BTC. *Btc* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://btc.com/>.
80. MINING POOL STATS. *Mining Pool Stats Ethereum* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://miningpoolstats.stream/ethereum>.
81. MINING POOL STATS. *Mining Pool Stats Monero* [online]. [N.d.] [visited on 2022-04-06]. Available from: <https://miningpoolstats.stream/monero>.
82. CAMACHO, José; WASIELEWSKA, Katarzyna. Dataset Quality Assessment in Autonomous Networks with Permutation Testing. *Seventh IEEE/IFIP International Workshop on Analytics for Network and Service Management*. 2022.
83. GOOGLE, LLC. *Classification: True vs. False and Positive vs. Negative* [online]. 2020 [visited on 2022-04-13]. Available from: <https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative>.
84. RASCHKA, Sebastian. An overview of general performance metrics of binary classifier systems. *arXiv preprint arXiv:1410.5330*. 2014.

Contents of enclosed CD

README.txt	file with contents description
└ datasets	
└ datasets.7zip	archive with datasets 01-04
└ notebooks	folder with Jupyter notebooks
└ rule_generator	folder with source codes of rule generator
└ src	
└ detector	folder with source codes of detector
└ aggregator	folder with source codes of aggregator
└ reporter	folder with source codes of reporter
└ thesis	folder with L ^A T _E X source codes
└ text	
└ thesis.pdf	thesis in PDF format