TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Anomaly Detection for Network Security based on Streaming Data

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Alexander Hartl, BSc.
Registration Number 01125115

to the Faculty of Electrical Engineering and Information Technology

at the TU Wien

Advisor: Univ. Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby
Assistance: Dr.techn. Félix Iglesias Vázquez, MSc.

The dissertation has been reviewed by:
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Prof. Dr. Marco Mellia

Vienna, 17th May, 2023

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Alexander Hartl, BSc.

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 17. Mai 2023

Danksagungen

Ich möchte meinen Dank an Prof. Tanja Zseby, aber auch an Joachim Fabini und Félix Iglesias ausrichten für eine engagierte und persönliche Betreuung, und auch für ein harmonisches und produktives Arbeitsumfeld, wie man sie anderenorts vergeblich sucht. Danke an Max, Fares, Bernhard und Lisa für viele kurzweilige Stunden im und jenseits vom Büro. Danke meiner Schwester, Trixi, und meinen Eltern für die Unterstützung auf so vielen Ebenen.

Abstract

Identifying attacks in network traffic constitutes a promising application area of Machine Learning (ML) and data mining techniques. While in related work many traditional ML techniques are presented with impressive detection performance under laboratory conditions, they show severe shortcomings and performance drops when implemented in real life. This can be explained when considering several challenges that data scientists in this area have to face. In particular, (a) traditional static models cannot cope with dynamics of network data, (b) model predictions often lack explainability, impeding successful deployability in practice, (c) systems that aim at detecting network attacks are faced with a highly adversarial environment, and (d) detectors developed in the past frequently relied on information that is not available for encrypted traffic. In this thesis, we address these challenges by developing novel methods for network traffic analysis and attack detection.

In particular, we investigate techniques appropriate for dealing with concept drift in the context of network traffic that allow continuous training throughout usage. We analyze algorithms suited for streaming anomaly detection, which are thus able to adjust to evolving characteristics of observed traffic, and present a new algorithm suited specifically for the high-speed requirements in data network environments. We propose and evaluate the use of visualization techniques for explainable ML in the field of network traffic analysis, which are applicable even when deploying opaque recurrent deep learning techniques, and we develop novel techniques for analyzing encrypted traffic.

The methods and approaches we outline in this thesis are highly relevant for network traffic analysis in high-security infrastructures due to the very specific combination of challenges in this field. However, there is a variety of other fields and application areas in data science to which our methods can be applied. With this thesis, we introduce new directions for future research, and we outline methods and algorithms to address the challenges that analysis of network traffic yields in modern times.

Kurzfassung

Die Erkennung von Angriffen in Netzwerkverkehr ist ein vielversprechender Anwendungsbereich für maschinelles Lernen (ML) und für Data-Mining-Verfahren. Während in bisherigen wissenschaftlichen Publikationen viele herkömmliche ML-Techniken mit beeindruckender Erkennungsleistung unter Laborbedingungen vorgestellt wurden, weisen sie bei der Umsetzung in der Praxis erhebliche Mängel und Leistungseinbußen auf. Dies lässt sich erklären, wenn man mehrere Herausforderungen betrachtet, denen sich Data Scientists in diesem Bereich stellen müssen. Insbesondere können (a) herkömmliche statische Modelle die Dynamik von Netzwerkdaten nicht bewältigen, (b) fehlt es den Modellvorhersagen oft an Erklärbarkeit, was den erfolgreichen Einsatz in der Praxis erschwert, (c) sind Systeme, die auf die Erkennung von Netzwerkangriffen abzielen, mit einer hochgradig gegnerischen Umgebung konfrontiert, und (d) stützen sich die in der Vergangenheit entwickelten Detektoren häufig auf Informationen, die für verschlüsselten Datenverkehr nicht mehr verfügbar sind. In dieser Arbeit widmen wir uns diesen Herausforderungen, indem wir neue Methoden zur Analyse von Netzwerkverkehr und zur Erkennung von Angriffen entwickeln.

Insbesondere untersuchen wir Techniken, die für den Umgang mit Concept Drift im Kontext von Netzwerkverkehr geeignet sind und ein kontinuierliches Training während der Nutzung ermöglichen. Wir analysieren Algorithmen, die sich für die Erkennung von Anomalien in Streamdaten eignen und sich somit an die sich verändernden Merkmale des beobachteten Verkehrs anpassen können, und stellen einen neuen Algorithmus vor, der speziell für die Hochgeschwindigkeitsanforderungen in Datennetzumgebungen geeignet ist. Wir schlagen den Einsatz von Visualisierungstechniken für erklärbares ML im Bereich der Netzwerkverkehrsanalyse vor und evaluieren diese, selbst wenn undurchsichtige rekurrente Deep-Learning-Techniken eingesetzt werden, und wir entwickeln neuartige Techniken zur Analyse von verschlüsseltem Netzwerkverkehr.

Die Methoden und Ansätze, die wir in dieser Arbeit vorstellen, sind für die Analyse des Datenverkehrs in Hochsicherheitsinfrastrukturen aufgrund der sehr spezifischen Kombination von Herausforderungen in diesem Bereich sehr relevant. Es gibt jedoch eine Vielzahl von anderen Bereichen und Anwendungsgebieten in der Data Science, auf die unsere Methoden angewendet werden können. Mit dieser Arbeit geben wir neue Impulse für zukünftige Forschung und skizzieren Methoden und Algorithmen, um den Herausforderungen zu begegnen, die die Analyse von Netzwerkverkehr in der heutigen Zeit mit sich bringt.

Contents

Publications

Parts of the following academic publications have been reused in this thesis:

- **Alexander Hartl**, Félix Iglesias, and Tanja Zseby. dSalmon: High-Speed Anomaly Detection for Evolving Multivariate Data Streams. In 16th EAI International Conference on Performance Evaluation Methodologies and Tools. ACM, 2023.
- **Alexander Hartl**, Joachim Fabini, and Tanja Zseby. Separating flows in encrypted tunnel traffic. In 21st IEEE International Conference on Machine Learning and Applications, pages 609–616. IEEE, 2022.
- **Alexander Hartl**, Joachim Fabini, Christoph Roschger, Peter Eder-Neuhauser, Marco Petrovic, Roman Tobler, and Tanja Zseby. Subverting counter mode encryption for hidden communication in high-security infrastructures. In The 16th International Conference on Availability, Reliability and Security, pages 1–11, 2021.
- **Alexander Hartl**, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for RNNs. In 2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService), pages 148–156, New York, NY, USA, 2020. IEEE.
- **Alexander Hartl**, Félix Iglesias, and Tanja Zseby. SDOstream: Low-density models for streaming outlier detection. In ESANN 2020 proceedings, pages 661–666, 2020.
- Maximilian Bachl, **Alexander Hartl**, Joachim Fabini, and Tanja Zseby. Walling Up Backdoors in Intrusion Detection Systems. In Big-DAMA '19, pages 8–13, Orlando, FL, USA, 2019. ACM.
- Félix Iglesias, **Alexander Hartl**, Tanja Zseby, and Arthur Zimek. Are network attacks outliers? a study of space representations and unsupervised algorithms. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 159–175. Springer, 2019.

The author has additionally (co)authored the following publications:

- Félix Iglesias, Tanja Zseby, **Alexander Hartl**, and Arthur Zimek. SDOclust: Clustering with Sparse Data Observers. In 16th International Conference on Similarity Search and Applications. Springer, 2023.
- Félix Iglesias, **Alexander Hartl**, Tanja Zseby, and Arthur Zimek. Outlier Detection in Streaming Data: A Comparison and Evaluation Study. In Expert Systems with Applications. Vol. 233.
- Joachim Fabini, **Alexander Hartl**, Fares Meghdouri, Claudia Breitenfellner, and Tanja Zseby. SecTULab: A Moodle-Integrated Secure Remote Access Architecture for Cyber Security Laboratories. ACM. 2021.
- Félix Iglesias, Denis Ojdanic, **Alexander Hartl**, and Tanja Zseby. MDCStream: Stream Data Generator for Testing Analysis Algorithms. In 13th EAI International Conference on Performance Evaluation Methodologies and Tools, pages 56–63. ACM. 2020.
- **Alexander Hartl**, Tanja Zseby, and Joachim Fabini. BeaconBlocks: Augmenting Proof-of-Stake with On-Chain Time Synchronization. In 2019 IEEE International Conference on Blockchain. IEEE, 2019.
- **Alexander Hartl**, Robert Annessi, and Tanja Zseby. Subliminal Channels in High-Speed Signatures. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA). 2018.
- **Alexander Hartl**, Robert Annessi, and Tanja Zseby. A Subliminal in EdDSA. 2017 International Workshop on Managing Insider Security Threats. ACM, 2017.

CHAPTER 1

Introduction

Before diving into our scientific work, we provide an overview of the thesis in this chapter. We introduce its research focus and motivate our research. We discuss our main research questions and outline and summarize our contributions prior to sketching the remaining thesis' structure.

1.1 Preface

A critical infrastructure provides services that are essential to several aspects of today's society like its economy, public health, safety or security [1]. Power distribution systems like the electrical power grid are frequently named as examples of critical infrastructures, not only due to their mere importance, but also because their transformation and adaptation due to the transition towards renewable energy sources yields various challenges for the society.

Information Technology (IT) provides various benefits for optimizing the operation of critical infrastructures and therefore is increasingly used. However, critical infrastructures are an attractive target for attackers due to political interests, potential for ransom or data exfiltration possibilities. Security of the IT systems used therefore is crucial and security breaches might have detrimental consequences.

Critical infrastructures can thus be considered a paramount example of high-security infrastructures, where protection of IT systems is a major concern. One part of a protection strategy is the detection of attacks, malware or unwanted behavior by using Network Intrusion Detection (NID). Data science yields various approaches that might be used for detection. However, many challenges need to be solved for successful and reliable deployment of data science methods for protecting high-security infrastructures:

(a) In contrast to many application fields of data science, the characteristics of network traffic require processing data as a stream where new data samples are seen steadily.

(b) To warrant good interaction of human experts with the Intrusion Detection System (IDS), a main requirement is that predictions of the used methods can be understood and interpreted, i.e. it is possible to explain why classifications have been made as they have been and what modifications of observed data would be necessary to change predictions.

(c) Furthermore, IDSs in high-security infrastructures are faced with a highly adversarial environment where sophisticated attackers use targeted attacks to evade detection. In such a scenario, the simple concept of matching observed data against known attack patterns is of limited use.

(d) Due to the increasing use of encryption, information of transmitted payload and protocol fields of encrypted network protocols are not available for IDSs.

In this thesis, we address all of these challenges and investigate techniques for constructing IDSs for use in high-security infrastructures. We therefore put emphasis on techniques that are interpretable by themselves or that allow to explain predictions made by ML classifiers. Due to the nature of the data that needs to be processed for NID, our discourse is focused on streaming data. We evaluate how known methods and our newly devised methods fit into high-security infrastructures and show inevitable limitations of IDSs when being used in this scenario.

1.2 Motivation

The problem of detecting attacks in network traffic has been an actively researched task for a long time and has immediate applications in practice. With high-security infrastructures becoming a target for such attacks, the relevance of IDS research has even risen.

The aspects we explore in this thesis are highly contemporary from multiple perspectives:

- IT is being increasingly used for the operation of critical infrastructures and it has become necessary to explore security-related questions in this context. Consequently, the time is now to approach these questions from an analytical and empirical perspective.
- The field of explainable and interpretable ML has become an active research area only recently. It therefore is natural to investigate these new methods in the context of IDSs.
- Due to the substantial amount of traffic that has to be processed in larger networks, computational efficiency is an aspect that has to be kept in mind when designing

IDSs. However, since computational power has increased to a larger extent than network capacity [2], we are now in the position to apply data science techniques to captured network traffic that would not have been possible before.

At the same time, we cannot load the task of devising appropriate methods in this field on practitioners: Due to the substantial importance for network security, methods must be carefully and comprehensively investigated and compared. Additionally, problem settings we explore in this thesis sometimes stand out from previous research in the field like, e.g., when it comes to flow-based analysis of encrypted Virtual Private Network (VPN) traffic. Exploration of these topics thus requires innovative ideas, which again require careful evaluation to yield good results.

In this thesis, we thus attend to the problem of attack detection in high-security infrastructures by investigating both, unsupervised methods for attack detection and supervised methods, putting an emphasis on explainability and interpretability. Our focus on anomaly detection methods for streaming data is due to network data being streaming data in their nature. Since general-purpose anomaly detection methods that have been introduced in the related literature cannot meet the specific characteristics of network data, we introduce novel methods that are specifically suited for application in this domain.

1.3 Research Questions

The application of ML techniques in NID is an area that still requires more research. To address the most pressing issues when aiming to deploy such principles in a high-security infrastructure, we aim to answer the following questions:

RQ1. Which state-of-the-art anomaly detectors are best suitable for detecting network attacks in streaming data?

The relevance of this question emerges due to the fact that with novel or very targeted attacks supervised methods might reach their limits. In contrast, in this question we target unsupervised anomaly detectors. Unsupervised methods might still be able to reveal the malicious traffic by detecting any traffic that deviates from normal behavior. Whether detection is possible needs to be measured by common metrics in the anomaly detection fields and evaluations need to be performed on state-of-the-art IDS benchmarking data. Commonly used metrics in this field are the ROC-AUC, AAP and AP@n, which we explain in Section 2.2.3. Suitable benchmarking data can be found in publicly available datasets, which we describe in Section 2.7. Besides providing reasonable detection results, a detection algorithm also needs to meet the characteristics of online Network Traffic Analysis (NTA). We can therefore state specific research questions as follows:

- What performance, measured in ROC-AUC, AAP and AP@n, can be obtained when evaluating state-of-the-art anomaly detectors for attack detection on suitable IDS benchmarking data?

- Which methods in the field of outlier detection are able to handle concept drift and are performant enough to meet the high-rate characteristics of network data?

RQ2. How can predictions of supervised network traffic classifiers be made explainable?

Explainability for supervised ML is an important and popular research area and, as described earlier in this chapter, highly relevant in our case. If we want to know the influence of certain feature values on a classifier's prediction, Partial Dependence Plots (PDPs) or Accumulated Local Effects (ALE) plots are state-of-the-art methods, which we will describe in Section 2.5. PDPs and ALE plots thus need to be evaluated in the context of NID. While these plotting techniques can directly be applied in many cases, modern Recurrent Neural Network (RNN)-based approaches use a feature structure that does not allow a straightforward application. To cover this important area of research, we thus need to explore:

- Can PDPs or ALE plots be used to detect classifier behavior for a given traffic pattern?
- How can the concept of PDPs be extended to the sequential setting?

RQ3. How can explainability be accomplished for unsupervised network traffic classifiers?

In the unsupervised domain, explainability to date has received less attention than in the supervised domain. Evidently, when deploying an IDS based on unsupervised techniques, it is equally important to explain and interpret classification outcomes. It is therefore important to evaluate anomaly detectors in this respect. For performing detection, but also for providing explanations, it is reasonable to use any information possible. A peculiarity that contrasts network data in this respect to many fields of anomaly detection is the importance of temporal information that is associated with network traffic. To illuminate explainability and interpretability for unsupervised ML, we therefore need to explore two main questions:

- Which methods for outlier detection provide interpretable results?
- How can an interpretable anomaly detector be designed that satisfies high-rate requirements and is able to cope with concept drift as explored in RQ1?

RQ4. Which implications does the use of extensive cryptographic techniques have for the attack surface in communication networks?

Encryption is considered a main pillow of modern information security. Yet, encryption adds additional complexity to the system and the fact that encryption does not exclusively

bring advantages for security, e.g. due to the possible exploitation of cryptographic methods for hidden information exchange, might come as a surprise for some. In particular, the use of encryption techniques that aggregate several connections on a common encrypted link is often thought to yield almost no possibilities for analysis of traffic, since traffic of very different types is mixed up. Since, however, this assumption has not been proven, we investigate it by attempting to separate individual flows in encrypted tunnel traffic. To assess the implications of encryption in high-security infrastructures we therefore arrive at three major questions:

- What threats can the use of cryptographic tools pose to high-security infrastructures?
- How is it possible to separate flows in encrypted VPN traffic and what quantitative separation performance can be obtained?
- What are suitable metrics for evaluating performance of successfully separating flows in encrypted VPN traffic?

1.4 Approach

We design our approach from the characteristics of high-security infrastructures. Hence, to be able to spot new or targeted attacks, we particularly investigate the feasibility of detecting attacks using unsupervised ML methods. Due to the nature of network traffic, a main building block for this task appear to be outlier detectors for streaming data. We will therefore evaluate to what extent known streaming outlier detectors meet the requirements of network traffic, improving them where necessary or devising new approaches. Also the feasibility of attack detection using unsupervised ML needs to be evaluated.

Encryption is a major building block of secure data transmission nowadays, but for NID applications it also has the downside that encrypted traffic can no longer be analyzed with ML techniques as efficiently as traffic submitted in clear text. While analysis of traffic encrypted on the transport layer has been investigated intensively already and in many cases only requires minor changes to the used ML techniques, the analysis of VPN tunnels provides challenges that have not yet been addressed. We will therefore explore whether it is possible to separate flows in VPN tunnels using ML techniques.

We design appropriate experiments to back our theoretical findings. This is on the one hand necessary to quantify detection performance that we can achieve using our investigated detection methods, on the other hand, since several approaches and scenarios that we describe are very novel, our experiments prove the practical feasibility of our designed algorithms.

Structuring the above tasks, we therefore schedule our work carried out throughout this thesis as follows:

1. To get a feeling for the applicability of outlier detection in the field, we first evaluate whether network attacks can be detected using outlier detection, and what detection performance can be achieved when basing experiments on available state-of-the-art methods for outlier detection.

2. As it turns out during state-of-the-art exploration, a main problem for the application of streaming outlier detection algorithms is that fast implementations are hardly available. We therefore turn to the more practical task of creating fast implementations of existing outlier detection methods.

3. Based on observed problems with state-of-the-art algorithms, we then construct specific algorithms that address the challenges of processing network data. In particular, we devise an algorithm that provides adequately low resource consumption and provides interpretability in particular considering the temporal nature of network traffic.

4. As explainability of ML is also crucial for supervised ML, we subsequently additionally investigate whether modern deep learning-based approaches for attack detection can be made explainable using PDPs or ALE plots.

5. Since high-security infrastructures use a high degree of traffic encryption, we finally have a more detailed look at the implications of using cryptographic techniques. As outlined above, we thus investigate whether it is possible to separate flows in encrypted tunnel traffic, but we also investigate the use of Galois/Counter Mode (GCM) encryption and show a novel method for hidden communication that would not be possible if traffic was not encrypted.

1.5 Contribution

With this thesis, we make a variety of contributions to the fields of both data science and network security:

- We perform a survey of outlier detectors for streaming data and contribute a framework providing efficient implementations of existing methods to be used in various fields of research.

- We show which combinations of outlier detection algorithms and feature vectors can be used for detecting network attacks based on a common IDS evaluation dataset. We show that attack detection using unsupervised methods is much more challenging than using supervised methods, as it has been done in related work.

- We design and implement a new outlier detector that is particularly suited for the characteristics of network data. We focus on known challenges in this field, i.e. detection performance, low memory and time complexity, interpretability and adaptability. However, we also introduce entirely new ideas by putting emphasis of

temporal patterns that naturally exist in network data, again providing benefits with respect to interpretability and accuracy.

- Besides our work on interpretability for unsupervised ML methods, we evaluate applicability of explainability methods to ML methods in the field of IDSs. We thus document findings related to the usability of explainability methods to discover unwanted classifier behavior, but we also develop adapted explainability methods suited for RNNs, which are particularly interesting in the field of IDSs.
- We develop an entirely new approach for the analysis of encrypted tunnel traffic, yielding the potential for enhancing capabilities for future IDSs.
- We present case studies for security challenges that come in conjunction with cryptographic techniques that have not been known before. For this, we discovered a new subliminal channel related to GCM encryption, and we devised novel methods for analyzing encrypted tunnel traffic. By depicting these challenges, we provide new directions for research and motivate the careful application of encryption techniques.

1.6 Structure

We begin our discussion by introducing important concepts and outlining related work that this thesis builds on in Chapter 2. In Chapter 3, we then investigate the potential of existing methods for unsupervised anomaly detection for detecting network attacks. Hence, besides benchmarking the potential of outlier detection algorithms for attack detection in terms of detection performance, we also take into account speed and resource usage considerations and develop a highly performant algorithm collection suited for application in NTA.

After investigating known algorithms, we devote Chapter 4 to the development of novel methods that enhance the state of the art by introducing an algorithm fitted specifically to the requirements we encounter in the analysis of network traffic. Besides developing outlier detection algorithms suited for this particular use case, we also consider the increasing use of encryption in modern communication networks and develop a method for analyzing tunnel traffic, which arguably constitutes the most potent encryption paradigm currently being used in practice.

However, also supervised methods provide valuable tools for IDSs and in many cases are able to come up with impressive detection performance for known attack patterns. To pave the way for successful deployment of popular supervised ML methods for IDSs, we investigate in Chapter 5 how their predictions can be explained and interpreted.

To complete our discourse, in Chapter 6 we depict further challenges a defender faces for IT security in high-security infrastructures by showing that malware communication patterns might be undetectable independent of which ML or data science is attempted for detection. We thus show that the methods investigated and devised in this thesis

can constitute an important part of a security strategy, but cannot be the only line of defense.

In Chapter 7, we discuss our previous findings and summarize their relevance for achieving and improving the security of high-security infrastructures. Chapter 8 concludes the thesis.

CHAPTER 2

Background

Research areas that we explore in this thesis are located in the intersection of information security and ML research. In this chapter, we provide the most important background and introduce the foundations of the methods we will use throughout the thesis.

2.1 Network Traffic Analysis

The field of NTA is concerned with extracting information from traffic that has been captured on an IP network link and classifying it. Data science yields various methods to approach this task. However, to apply modern methods for data analysis, network traces, which in most cases are captured in the form of Packet Capture files (PCAPs), have to be transformed into a more structured form. We outline these preprocessing steps in this section.

2.1.1 Defining Flows

For many ML tasks like classification or regression, it is necessary to transform the data into a form that allows assigning labels or desired prediction outcomes to individual instances. Hence, when processing network traffic, the question arises how this transformation can be done and what is represented by individual instances. Arguably, packets might be considered a reasonable smallest unit that makes up communication in packet-switched networks. Indeed, research has successfully applied classification methods on a packet level. However, in general individual packets are unlikely to provide sufficient information to identify the type of traffic unless one uses Deep Packet Inspection (DPI), i.e. additionally analyses the payload of packets. With today's ubiquitous traffic encryption, relying on the availability of payload information is an unrealistic assumption.

With individual packets failing to provide sufficient information, we thus need to split up the observed sequence of packets into smaller units of sequences that each can individually

be considered to serve a particular purpose and, hence, can be jointly processed. These units are termed *flows*. Hence, when assigning labels about which type of traffic is observed or whether it constitutes attack or benign traffic, such labelling is performed on a per-flow basis. Flows are typically based on proximity in time and on common properties like, e.g., a common source host or a common transport layer address, where the set of considered common properties is termed the *flow key*. The definition of flows necessarily needs to be held very flexible to account for a wide range of possible approaches used in NTA. IPFIX [12] defines the term flow as "*as a set of packets or frames passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties*" [12].

A very popular flow definition is the *5-tuple* flow key, where packets are grouped into flows by considering transport layer protocol, source and destination IP addresses and transport layer port numbers. The relevance of the 5-tuple flow key arises from the fact that in most cases it maps individual TCP connections and UDP streams to flows. Arguably, TCP connections and UDP streams can be considered as building blocks of modern network communication.

2.1.2 Feature Extraction

Notice of adoption from previous publications (Section 2.1.2)
Parts of the contents of this section have been published in the following paper:

> *[131] Félix Iglesias, Alexander Hartl, Tanja Zseby, and Arthur Zimek. Are network attacks outliers? a study of space representations and unsupervised algorithms. In* Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 159–175. Springer, 2019*

Text in Section 2.1.2 has partially been composed in collaboration with Félix Iglesias.

Flows typically consist of a sequence of network packets with inhomogeneous lengths, where each packet in a flow yields a number of features like, e.g., the packet's length or its Inter-Arrival Time (IAT) to the previous packet in the flow. However, many methods in ML and data science require data samples to be provided as a constant-length vector of features, disallowing a direct application on a flow's packet features. It is therefore necessary to aggregate features of packets contained in a flow into a static-size set of flow features, which are designed to yield as much information about contained packets as possible. Besides broadening the range of usable ML methods, this aggregation yields the additional benefit of reducing the resources required for storage and transfer, as extracted flow features typically occupy a substantially smaller amount of space than the concatenation of underlying packet features. Typically, data reduction is performed based on statistical values like the mean, the median or the standard deviation.

Several feature vectors have recently been proposed and used in the NTA literature:

- **CAIA vector**. As coined in [166], we use the term CAIA to refer to the feature vector originally proposed by Williams et al. [230]. The same vector has been commonly applied (as defined or with minimal variations) in the context of NTA, specifically when using ML-based solutions, e.g., [226] [240] [154]. The original CAIA vector stores bidirectional information and consists of 22 features. We extended it to 30 features as in [154].
- **Consensus vector**. In [85] a set of features for NTA is selected based on a meta-study including 71 of the most relevant, cited papers in NTA. This work concludes with 12 relevant features. We extend them based on the considerations discussed in [85] and [166], obtaining a final 20-feature vector.
- **Cisco-Joy vector**. Anderson et al. recently proposed this feature vector, which is able to discriminate attacks in supervised learning and is suitable for encrypted traffic [30] [31]. It contains 650 features and can be easily extracted by using the Cisco/Joy open tool[1].
- **Time-Activity vector (TA)**. The Time-Activity vector [129] uses a 3-tuple or 5-tuple key and is unidirectional. It was devised to profile flows from a time-behavioral perspective, allowing lightweight characterization of traffic by means of clustering methods. The final vector is formed by 13 features.
- **AGM vector**. Designed for the discovery of patterns in the Internet Background Radiation [130], this vector allows profiling traffic sources or destinations. The *basic AGM vector* contains 22 features, which are extended after removing nominal features or transforming them into dummy variables if distributions are concentrated on few values (e.g., more than 90% of traffic uses TCP, UDP or ICMP). The *extended AGM vector* is purely numerical.

The CAIA, Consensus, TA, and AGM vectors are compared in [166] for supervised attack detection with the UNSW-NB15 [171] dataset.

2.2 Network Intrusion Detection

There is a multitude of approaches for constructing IDSs. Not only are there a vast number of possibilities for creating taxonomies for IDSs, but also terms for describing methods are used inconsistently. For an overview of methods, in this section we distinguish the class of Signature-based Intrusion Detection System (SIDS) from ML-based systems. This separation follows the taxonomy in [143]. However, while ML-based systems are referred to as "anomaly-based systems" in [143], in this thesis we use a stricter definition of the term "anomaly" and reserve "anomaly" for unsupervised methods that detect anomalies and outliers in an algorithmic way.

[1] https://github.com/cisco/joy

2.2.1 Signature-based Systems

We provide a brief summary of SIDS from [143]. SIDSs are based on matching patterns in observed traffic with a database of previously known intrusions. Hence, when performing intrusion detection based on single packets, rules for features observed in these packets are composed to form conditional statements for when to raise an alarm. More complex rules able to process a sequence of packets might be implemented using state machines, formal language string patterns or semantic conditions.

For known types of intrusions, SIDSs provide several benefits like high performance in both detection accuracy and processing speed. The main problem of SIDSs lies in their limited capability of detecting zero-day attacks or attacks that deviate from previously known attacks, e.g., because it is launched by polymorphic malware.

The probably most well-known SIDS application is SNORT [191].

2.2.2 Machine Learning-based Systems

ML techniques are used or tested in many application areas nowadays. In the context of network data, an ML-based IDS promises to extract traffic characteristics from available network data in an automated way and build a model from it. Hence, in contrast to signature-based systems the expensive procedure of devising appropriate rules is avoided. However, data is needed for training classifiers, which in many situations is difficult to collect.

Two commonly used paradigms in ML are *supervised learning* and *unsupervised learning*. In supervised learning in NID, training data consists of traffic traces for each traffic class that should be detected during later operation. Hence, the respective traffic traces are labeled, i.e. each packet can be associated to the respective class. A multitude of different methods for supervised ML have been devised. In the context of NID, tree-based methods have shown to achieve eminent detection results while being reasonably resource-efficient. Fueled by their popularity, also Neural Networks (NNs) have been evaluated in the context of NID and showed similarly good results.

Unsupervised learning describes methods that do not require the data to be labeled. A well-known class of algorithms performing unsupervised learning are clustering algorithms, which aim to find a partitioning in the provided data that reflects the true class separation. Another class of unsupervised learning methods are outlier detection or anomaly detection methods. Outlier detection methods aim to detect samples within the provided dataset that deviate from all remaining samples by violating the data's major shapes in feature space. A variety of outlier detection algorithms have been proposed, which differ in how an outlier is defined. Hence, it is not clear whether the samples that are exceptional in an intuitive sense meet the outlier definition of an algorithm to be deployed.

2.2.3 Evaluating IDSs

Despite the multitude of different concepts for constructing IDSs, it is illusive to expect that an IDS can perfectly separate between benign traffic and attack traffic. Hence, the question emerges how to compare different methods for constructing IDSs to select the most suitable one.

Indeed, even this question of how to ideally evaluate an IDS is not trivial to answer, since it heavily depends on the impact that misclassified traffic samples have, but also on the type and frequency of occurring attacks. In the IDS domain, we frequently deal with strongly imbalanced datasets, where benign traffic samples are much more frequent than attack samples. It is a common misconception that the problem of dataset imbalance can simply be alleviated by switching to specific alternative evaluation metrics. In fact, to perform realistic evaluation, it would not only be needed to compare to what extent attack sample proportion occurring during application is represented by the evaluation dataset, but it would also be necessary to quantify to what extent misclassification of attack samples is more expensive than misclassification of benign samples. Since this information usually is not available, we have to revert to reporting readings for a selection of different evaluation metrics to provide a good overview of detection performance.

In general, since our goal is performing only binary classification, we can revert to established metrics from ML research. Adhering to common practice for binary classification problems, we partition an evaluation dataset into four disjoint sets we will denote as follows:

True Positives (TPs) denote **attack** samples that are **correctly** classified,
True Negatives (TNs) denote **benign** samples that are **correctly** classified,
False Positives (FPs) denote **benign** samples that are **wrongly** classified and
False Negatives (FNs) denote **attack** samples that are **wrongly** classified.

Adopting common practice, we will denote by TP, TN, FP and FN also the respective sets' sizes to keep notation concise.

Furthermore, we have to distinguish whether our evaluated classifiers report a real-valued score for a sample being an attack or only report a binary label. In what follows, we will elaborate on the most important metrics used in this thesis.

Binary Predictions

Even in the simplest case where binary classification is performed with a Boolean classifier output, several metrics can be derived and are frequently used. An important example is **accuracy**. Accuracy denotes the proportion of correctly classified samples within the entire evaluation dataset. Hence,

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{2.1}$$

In the field of NID, missed attacks can have detrimental effects, while erroneously classified benign traffic samples can be inspected manually and thus have less extensive implications.

To evaluate only the proportion of reported attack samples among the entirety of attacks while neglecting the impact of false positives, we can use **recall**. Recall is computed as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.2}$$

On the other hand, specific ML applications might require the reported positives to be as free of false positives as possible, while not putting too much weight on reporting all possible samples. In this case, **precision** is an important metric, which is computed as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.3}$$

and, hence, measures how clean the reported set of positives is.

In many cases, precision can be increased at the cost of recall and vice versa. Intuitively, it is thus desirable to optimize both, precision and recall. Two approaches that come to mind to consider both, recall and precision, is two either report the minimum of the two metrics or to use the mean value of both. The **F1 score** represents an intermediate way between both these approaches by reporting the harmonic mean between recall and precision, since the harmonic mean of two non-negative, real-valued numbers always lies between their minimum and their arithmetic mean. The F1 score is thus computed as

$$\text{F1} = \frac{2}{1/\text{Precision} + 1/\text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}. \tag{2.4}$$

Accuracy has the downside of providing misleading readings when working with strongly imbalanced data. For instance, in a dataset with binary labels where 99% of samples belong to the negative class, a degenerate classifier that always predicts the negative label would obtain 99% accuracy. A metric that does not suffer from this shortcoming is **Youden's J statistic**, which was introduced in [237] and is computed as

$$\text{J} = \frac{\text{TP}}{\text{TP} + \text{FN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} - 1. \tag{2.5}$$

A classifier that performs random guessing is assigned a Youden's J statistic of 0, while a perfect classifier is assigned a value of 1. Negative values are possible, in which case the classifier's predictions should be flipped.

Real-valued Predictions

Many classifiers output a real-valued prediction for a flow to be an attack. Even though in many cases this value cannot directly be interpreted as probability value, we can order flows for their attack likelihood. A real-valued classification value can always be converted to binary prediction by introducing a threshold value that defines the minimum prediction value for a sample to be classified as attack. Since this thresholding operation leads to a loss of information, IDS evaluation directly from real-valued outputs might be

considered a more truthful approach, in particular if it is not clear how to choose the threshold value. Here, we summarize the most important metrics for this purpose from [63].

A metric that is often used in research is derived from the Receiver Operating Characteristic (ROC), which plots the TP rate against the FP rate when varying the threshold. The desirable scenario of achieving a high TP rate and, at the same time, a low FP rate corresponds to a high area under the resulting curve. This **Area under the ROC curve (ROC-AUC)**, which is also sometimes abbreviated as AUC, can therefore be used for evaluating a classification result. An interesting property of the ROC-AUC is its probabilistic interpretation. In particular, it can be shown that the ROC-AUC corresponds to the probability of ranking an attack flow that has been randomly picked from the evaluation dataset higher than a randomly picked benign flow.

Another important metric is the **Precision at n (P@n)**, which measures the precision of reported attacks when considering the top n places. Hence, it is computed as

$$P@n = \frac{|\{a \in A | \text{rank}(a) \leq n\}|}{n}, \tag{2.6}$$

where A denotes the set of all positives, i.e. all attacks, and $\text{rank}(a)$ denotes a's position when sorting all samples in the dataset according to the reported scores in descending order. Hence, in this definition n is a parameter, which has to be set beforehand. A popular and natural choice is setting n to the number of positives, i.e. $n = |A|$. However, since the number of positives is not known in application, we can use the obtained positions of attack samples for n, averaging over all attacks samples. This metric is known as **Average Precision (AP)** and is computed as

$$\text{AP} = \frac{1}{|A|} \sum_{a \in A} P@\text{rank}(a). \tag{2.7}$$

Both P@n and AP suffer from dataset imbalance. To transform them to yield a value of 0 for random predictions, adjustment for chance can be applied. Campos et al. [63] provide the relations

$$AP@n = \frac{P@n - |A|/N}{1 - |A|/N} \text{ and} \tag{2.8}$$

$$\text{AAP} = \frac{\text{AP} - |A|/N}{1 - |A|/N} \tag{2.9}$$

with the total number of samples in the dataset N for the **Adjusted Average Precision at n (AP@n)** and the **Adjusted Average Precision (AAP)**, respectively.

2.3 Machine Learning Methods

A vast number of methods have been proposed in the context of ML. In this thesis, we focus on Decision Trees (DTs), Random Forests (RFs) and NNs, which have shown

good performance in the context of NID. Additionally, the field of unsupervised anomaly detection is highly relevant for this thesis.

2.3.1 Decision Trees and Random Forests

An important family of classification methods we use in this thesis is that of DTs and RFs, which have shown good performance in the context of NID.

Not only in data science and ML, but also in other domains, constructing a DT constitutes a popular method of arriving at decisions. A DT is a tree-shaped sequence of criteria that are sequentially tested on the available information, until a leaf is reached, which provides the desired decision.

Hence, in the field of data science, a DT can be considered a number of `if` conditions tested on the provided input feature vector that are arranged in a tree-like structure. While the classification process for a DT is markedly simple, it is the training process that defines the behavior and performance of the resulting classifier. In this thesis, we use the probably most popular training procedure, which involves building the tree from the top to the bottom where at each branch the most suitable split is chosen by maximizing a certain criterion. This criterion is the Gini impurity of the resulting dataset partitions. To counterfeit overfitting, DTs are usually not split until leaving only single samples at the leaves, but a stopping criterion is defined like the maximum depth or the minimum number of samples within one leaf. In data science research, scikit-learn [179] is the likely most popular framework for developing DT and RF classifiers. We refer to scikit-learn's documentation[2] for an overview of popular measures for regularization.

To optimize prediction performance, DTs are frequently used in an ensemble, resulting in RF classifiers. In this case, a multitude of DTs are independently trained and their predictions are averaged for the RF's final prediction. Variety in training of DTs is achieved by feeding different randomly selected portions of the training dataset to the individual DTs.

2.3.2 Neural Networks

Gradient descent is an efficient and popular method for finding a local extremum of a differentiable function: By iteratively taking steps in the direction of largest descent, input parameters of the function can be found that minimize that function's output.

Training using gradient descent can be considered a main characteristic of Neural Networks (NNs). In particular, for NNs a function is constructed as composition of simple mathematical functions, so that the resulting function is differentiable in its parameters, but at the same time is able to express highly non-linear relations.

A simple NN is depicted Figure 2.1. Each circle in this figure depicts an affine mapping in the indicated inputs, combined with an *activation function*, i.e. a non-linear function

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

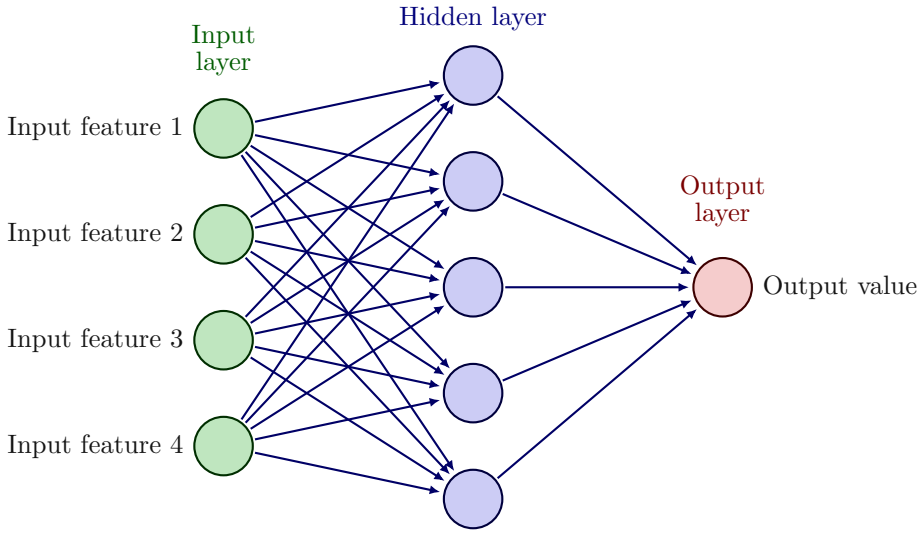


Figure 2.1: A simplistic NN, as it might be used for binary classification.

of the resulting scalar output. Weights of all affine transformations are the parameters that are determined during NN training using gradient descent.

Hence, the desired function is constructed in multiple layers to be able to capture a sophisticated behavior. The Multilayer Perceptron (MLP) depicted in Figure 2.1 can be considered the simplest example for NNs, where input values conform to a simple feature vector and output values can, e.g., denote attack/non-attack classification.

Autoencoders A further simple and frequently used type of NN is the autoencoder. An autoencoder is an MLP, where the NN's input dimension is as large as its output dimension and the NN is trained to output the same values as provided at the input. However, at an intermediate layer the number of neurons is chosen substantially smaller than the input's dimension, so that input values cannot simply be passed to the output as they are. Instead, NN training is supposed to find dependencies and structure in the data, so that the input can be encoded into a vector of lower dimension and the original vector can be recovered with small error. Consequently, the input-facing part of the NN is frequently called the *encoder* network, while the output-facing part is called the *decoder* network.

Autoencoders might be used for dimensionality reduction, e.g., for compression purposes or as a preprocessing step for other data mining techniques. Another purpose of autoencoder training might be running training as proxy task that does not require labeled data, so that a trained NN can be used as basis for transfer learning of a different task.

Recurrent Neural Networks In NID, when not extracting statistical features from flows, features have a sequential nature, since a flow consists of a sequence of packets,

where features can be extracted from each of the packets. An RNN denotes an architecture of NNs that is able to process sequential data. Here, packet feature values are sequentially fed into the NN from the first packet to the last packet. Additionally, recursive connections are added by recurrent units, so that information is passed on from step to step and the NN is theoretically able to make a decision involving all packet features in the final step.

2.3.3 Anomaly Detection

IDSs using anomaly detection or outlier detection allow identifying attack samples as samples that set themselves apart from the majority of network traffic. It is not trivial to define the term "outlier" or "anomaly". Hawkins [122] has provided a popular definition in colloquial speech: An outlier is *"an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism"* [122]. Since this colloquial statement cannot be used to perform algorithmic detection of anomalous samples, a wide range of algorithms have been developed to capture outlierness in a static dataset. Using a coarse and non-comprehensive classification, many traditional outlier detection algorithms work using either

- a **distance-based** definition of outlierness. Taking the most simple outlier detector as example, the k Nearest Neighbors (kNN) rule defines the distance to the k^{th} nearest data point as outlier score, thus declaring points lying in a densely populated neighborhood as inliers.
- a **density-based** definition of outlierness. Since the classical kNN rule suffers from the problem of varying outlier scores with cluster density, the Local Outlier Factor (LOF) [58] measures outlierness with respect to neighbor distances of nearest neighbors.
- a **histogram-based** definition of outlierness. Intuitively, outliers lie in low-probability regions in feature space, giving rise to a probability-based definition of outlierness. As a simple example, the Histogram-based Outlier Score (HBOS) [101] bases the reported outlier score on the bin width of univariate histograms of all features.
- **clustering algorithms** to declare points that cannot be matched to any cluster as outliers. Since outliers might deteriorate a clustering algorithm's functioning, many such algorithms have built-in means to ignore outlying points as noise.
- or the **reconstruction error** of an autoencoder to determine outlierness. Since autoencoders are trained to describe the features of a data sample with a low-dimensional encoding vector, the reasoning behind using the reconstruction error is that reconstruction fails when a data sample's structure does not match the one observed in usual data. Hence, anomalous samples are supposed to be assigned a higher reconstruction error than normal data.

This list is only meant to provide a simple overview in popular methods. In fact, due to the wide variety of different algorithms, many work by means that cannot clearly be matched to one of these categories. In most cases, outlier detection algorithms report a real-valued outlier score instead of just a binary label to indicate outlierness.

2.4 Stream Data Processing

In their nature, data samples from network traffic arrive as a stream of data. E.g., whenever a network flow finishes, the flow's feature vector becomes available and should be processed by an IDS as quickly as possible and a decision should be provided as soon as possible. This is in contrast to many more traditional ML applications like in visual computing or speech recognition, where a dataset is provided for training and later evaluations only need to be performed against the trained model.

A main problem in a setting using streaming data is data volume. A rule-based IDS likely is able to capture newly arriving data samples at a high rate. However, creation of a dataset for training an ML classifier requires storing samples that arrived during a long-enough time period. If samples arrive at a high rate, the accumulative volume of the resulting dataset can become considerable or might even become prohibitive in some scenarios.

Furthermore, another problem in a streaming scenario is *concept drift*. Concept drift captures the fact that observed data changes throughout time as, e.g., users change their behavior. Hence, to capture an accurate notion of normality, the model must be retrained with updated data or the algorithm must allow to continuously update the model as new samples arrive.

An exemplary simple, yet popular approach to continuously adapt a model as new data arrives is the use of a Sliding Window (SW). For SW techniques, the most recent samples are kept and used as model for evaluating new data, but samples are dropped and removed from the model as soon as they become too old. For example, a simple kNN algorithm might be implemented by assessing a newly arriving sample based on the k nearest neighbors within a SW of the most recent N points, with $k, N \in \mathbb{N}$ being constants. We refer to the later Chapter 3 for a comprehensive overview of methods to handle concept drift.

2.5 Explainable AI

Notice of adoption from previous publications (Section 2.5)
Parts of the contents of this section have been published in the following paper:

> *[116] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling Up Backdoors in Intrusion Detection Systems. In* Big-DAMA '19*, pages 8–13, Orlando, FL, USA, 2019. ACM*

Text in Section 2.5 has been composed in collaboration with Maximilian Bachl.

Different ML methods show varying qualities with respect to explaining and interpreting predictions and decisions the model provides. For instance, a single DT of small size can be considered as a sequence of simple yes/no questions and thus easily be interpreted. As the tree's size increases or when combining a large number of trees in a RF, interpretability fades. NNs allow interpretation only in the simplest cases, especially when degenerating a NN to a linear regressor.

Explainability methods thus aim to provide explanations for ML models that are not directly explainable by themselves. A frequently used example is making use of a surrogate model, which operates by training a simple, explainable ML model in the neighborhood of the data sample, thus allowing evaluating how features influence the classifier's prediction for the specific data sample.

In this thesis, we focus on visualization techniques for explaining model predictions. By providing a visual representation, such techniques yield the potential to give a very intuitive and comprehensive overview of how certain features influence the prediction outcome. However, if the number of features is high and model behavior is complex, providing graphical representations that depict actual model behavior is difficult. In this thesis, we focus on the techniques of PDPs and ALE plots.

2.5.1 Partial Dependence Plots

PDPs were proposed in [92] and visualize dependence of a model's predictions by plotting the Model under Investigation (MuI)'s prediction for a modified dataset for which the feature's value has been fixed to a certain value, averaging over the modified dataset.

If we denote by $\boldsymbol{X} \in \mathbb{R}^n$ a random vector drawn from the feature space and by $f(\boldsymbol{X}) \in [0, 1]$ the prediction function, the PDP for the i^{th} feature X_i can be expressed as

$$\text{PDP}_i(w) = \mathbb{E}_{\boldsymbol{X}}\Big(f(X_1, \ldots, X_{i-1}, w, X_{i+1}, \ldots X_n)\Big). \tag{2.10}$$

Empirically, we can approximate the distribution of the feature space using the distribution of observed samples. Hence, at a given point w, the PDP for the i^{th} feature can be found by setting the i^{th} feature value in all samples in the dataset to w and averaging over the predictions of the resulting modified dataset.

2.5.2 Accumulated Local Effects

In real situations, datasets usually exhibit a non-negligible degree of feature dependence. Due to feature dependence, areas exist in the feature space that are unlikely to occur. Since a model is trained with real, observed data, the training set therefore might not include samples for these areas. Consequently, the model's predictions become indeterminate for these areas, posing a problem when considering these predictions for computing PDPs.

In an attempt to overcome this problem, it is possible to only consider samples which are likely to occur for certain feature values, i.e. to consider the conditional distribution of remaining features for computing explainability graphs.

ALE plots [34] make use of this idea. For the i^{th} feature X_i, the ALE plot $\text{ALE}_i(w)$ can be defined differentially as

$$\frac{d}{dw}\text{ALE}_i(w) = \mathbb{E}_{\boldsymbol{X}|X_i}\left(\frac{\partial}{\partial X_i} f(\boldsymbol{X}) \,\middle|\, X_i = w\right). \tag{2.11}$$

To combat ambiguity of this definition, we force $\text{ALE}_i(w)$ to have zero mean on the domain of X_i. For empirical evaluation, we approximate the conditional distributions of $\boldsymbol{X}$ by averaging over samples for which $X_i \approx w$. In this thesis, we used the 10 closest samples for estimating the distributions.

2.6 Specific Requirements of High-Security Network Infrastructures

Communication networks are becoming increasingly important for critical infrastructures. It is the tremendous importance of this type of infrastructures that gives rise to their high demands in network security. For our current discourse, these demands lead to more characteristics than just motivating the use of a potent IDS. In what follows, we elaborate on these characteristics.

2.6.1 Trust in ML-based Decisions

If the stability of a large critical infrastructure depends on proper detection of attacks, but also on avoidance of false positives, it is not sufficient to obtain a final classification from a classifier. Instead, persons in charge need to know which properties and characteristics have led to a specific flow being considered an attack or non-attack to judge whether the behavior is reasonable. Only if it is possible to understand why predictions have been performed the way they have been, trust in the respective classifier can be established, paving the way for successful deployment in a live system.

This requirement for trust in an ML-based system therefore implies either the use of an explainable ML model or the use of techniques that allow explaining and interpreting predictions of ML models that would otherwise not be explainable.

Table 2.1: Datasets used throughout this thesis.

Dataset	Description	Year
CIC-IDS-2017 [203]	Labeled synthetically generated network traces	2017
UNSW-NB15 [170]	Labeled synthetically generated network traces	2015
MAWI [69]	Unlabeled network traces	2021
KDD Cup'99 [4]	Labeled network-based and host-based IDS data	1999
SWAN-SF [33]	Solar flare measurements	2020
Patch Tuesday [62]	Darkspace network traces	2012
Thyroid Disease [13, 63]	Medical data on hypothyroidism	1987
Cardiotocography [13, 63]	Medical data on heart diseases	2000
Page Blocks [13, 63]	Separate text from images in documents	1995

2.6.2 Network Architecture

To achieve superior security, critical infrastructures frequently incorporate security considerations already in the design phase and in the layout of the network architecture. For instance, to impede communication between arbitrary network participants, strongly segmented network architectures are deployed that use firewalls or application layer gateways to only allow necessary communication. Furthermore, specific hardware with hardened security properties is frequently used for performing the most sensitive cryptographic operations to securely store secret keys and thus gain security benefits.

2.6.3 Attack Sophistication

Attacks on critical infrastructures that target at ransom or are motivated by political interests can be backed by considerable financial resources. They therefore can hardly be compared to network attacks targeting enterprise networks of typical companies. For instance, Advanced Persistent Threats (APTs) play an important role for security considerations, i.e. attacks that infiltrate the network with malware that stays undetected for a long period. Related to this kind of attack, but also as a general technique to evade detection, also techniques for hidden communication between malware are relevant. Techniques for hiding information in not fully used network protocol fields are called *covert channels*.

2.7 Datasets

Since experimental evaluation is a main method used throughout this thesis, datasets are required to perform evaluation in a realistic fashion. Several requirements have to be met for a dataset to allow realistic performance benchmarks. Algorithms investigated in this thesis are not limited to processing network data and, since we investigate several research tasks in the thesis, we use several datasets with varying characteristics. We show an overview of used datasets in Table 2.1.

Our main focus being network data, we require a labeled datasets providing network data for most of our evaluations. We used the CIC-IDS-2017 [203] and UNSW-NB15 [170] datasets to meet these characteristics.

The CIC-IDS-2017 [203] dataset was created by the Canadian Institute of Cybersecurity (CIC), as they found all available datasets for NID tasks to have substantial shortcomings. A dataset for evaluating IDSs has to be complete, realistic, representative, diverse and heterogeneous with respect to protocols, attacks, legitimate uses and formats, to provide realistic performance measurements [97]. The provided network captures yield more than 2.3 million flows when preprocessed based on a 5-tuple flow key, and contain legitimate traffic and attack traffic from botnets and DoS, infiltration, brute force, web attacks and scanning attacks. The dataset is freely available on the Internet, allowing reproducibility of our results. However, similar to other IDS evaluation datasets, shortcomings have been reported for CIC-IDS-2017 [80]. We therefore avoided using preprocessed data from the dataset directly. Instead, we used and improved tools and scripts for flow extraction and labeling that are maintained by the CN Group at TU Wien [164, 227], and carefully investigated the resulting labeled data.

As a second dataset for evaluating NID results, we used the UNSW-NB15 [170] dataset, which was created by researchers of the University of New South Wales (UNSW). It contains over 2 million flows when using a 5-tuple flow key and consists of benign traffic and various types of attacks, together with a ground truth file. Malicious traffic contained in UNSW-NB15 includes reconnaissance, DoS and analysis attacks, exploits, fuzzers, shellcode, backdoors and worms. Similar to CIC-IDS-2017, the UNSW-NB15 dataset can be obtained freely from the Internet and preprocessing tools we used can be obtained from [164, 227] to reproduce our results.

Attacks contained in CIC-IDS-2017 and UNSW-NB15 are shown in Table 2.2.

Table 2.2: Flow occurrence frequency of attack types.

(a) CIC-IDS-2017

Attack type	Proportion
DoS Hulk	10.10%
PortScan, Firewall	6.90%
DDoS LOIT	4.08%
Infiltration	3.30%
DoS GoldenEye	0.32%
DoS SlowHTTPTest	0.18%
DoS Slowloris	0.17%
Brute-force SSH	0.11%
Botnet ARES	0.03%
XSS attack	0.03%
PortScan, no Fw.	0.02%
Brute-force FTP	0.01%
SQL injection	<0.01%
Heartbleed	<0.01%

(b) UNSW-NB15

Attack type	Proportion
Exploits	1.42%
Fuzzers	1.01%
Reconnaissance	0.58%
Generic	0.21%
DoS	0.19%
Shellcode	0.08%
Analysis	0.03%
Backdoors	0.02%
Worms	0.01%

CHAPTER 3

Applying Stream Outlier Detection to Network Data

As a potentially crucial building block for a potent IDS, we first explore to what extent unsupervised methods can be used for attack detection. In the light of the current state of the art, our work needs to focus on two domains to explore this question:

1. Due to the streaming nature of network traffic, streaming outlier detection techniques are a good candidate for unsupervised attack detection. Several methods for outlier detection on streaming data have been proposed in literature, prompting the consolidation of available methods. Due to the large amount of traffic data that needs to be processed in this application area, a main problem concerns implementation efficiency, which needs to be addressed.

2. Furthermore, it is not yet established to what degree network attacks can be considered outlying data samples and, hence, can be discovered with outlier detection methods. The success of this task might not only depend on which outlier detection method is used, but also on the used feature vector, requiring a comprehensive evaluation of different approaches for constructing an unsupervised IDS.

In the following sections, we thus explore outlier detection techniques with respect to both aspects. In terms of the research questions we have presented in Section 1.3, in this chapter we thus mainly focus on RQ1. Naturally, a main focus is the evaluation of state-of-the-art methods with respect to the performance metrics we have outlined, which requests us to perform suitable experimental evaluations, but also to develop suitable algorithm implementations as a basis for these experiments. We also analyze the functional building blocks of established algorithms to illuminate their suitability with

respect to the ability to handle concept drift and with respect to high-rate streaming settings.

We thus begin by surveying outlier detection techniques for streaming data in Section 3.1. Since in most cases algorithm implementations are not available that are efficient enough for meeting the demands in this application area, we continue by designing and implementing a framework that allows efficient use of existing methods in Section 3.2. We then evaluate in Section 3.3 the combinations of several feature vectors and outlier definitions to investigate the feasibility of attack detection with unsupervised methods.

3.1 A Comparison of Stream Outlier Detection Techniques

Data streams have to be processed in many applications and occur in various forms. The well-known problem of spotting anomalies in streaming data processing therefore similarly is an ambiguous one, which has been researched in many different aspects and facets.

One particular kind of stream that occurs, e.g., in monitoring applications, in fraud detection, IDS research, but also in medical applications, are streams where independent multivariate data samples arrive steadily at a non-constant arrival rate. A highly relevant anomaly detection task in this setting is detecting individual outlying samples. In an IDS application, such samples might correspond to, e.g., malicious network traffic that ought to be detected and blocked.

Several methods have been proposed for anomaly detection in such a setting, which partially adopt known techniques from static anomaly detectors, but partially also devise new approaches that are optimized for the streaming nature of such processes. However, literature lacks a comprehensive overview of such methods for anomaly detection.

In this section, we thus survey outlier detection methods for streaming data. We provide a systematic overview of basic building blocks of the corresponding algorithms, treating the method for handling concept drift and the basic outlier detection method separately. Besides discussing and comparing existing approaches, our overview thus allows to identify new combinations of established methods and to construct new algorithms that are customized to specific use cases.

3.1.1 Functional Principles of Outlier Detection Algorithms

We can distinguish two main aspects about the functioning of outlier detection algorithms for streaming data, relating to spatial and temporal aspects of algorithm operation.

1. On the one hand, a method for anomaly scoring is needed, which provides the definition of outlierness. This part corresponds to the functioning of traditional outlier detection algorithms for static data and in many cases adopts their concepts.

Table 3.1: Approaches for handling concept drift.

	$O(1)$	Cut-off for old data	Adaptation to new data
SWs		✓	Very fast
Reference Windows (RWs)	✓	✓	Slow
Exponential moving averaging	✓		Fast

2. Unlike algorithms operating on static data, however, for streaming data it is additionally needed to consider concept drift and adapt the model for continuously arriving newly seen data.

In many cases, a simple combination of methods for concept drift and outlier detection is used and both approaches can clearly be separated. Hence, one method is used for continuously updating a model, which is then used as basis for outlier detection.

Approaches for Handling Concept Drift

During the operation of an outlier detector the characteristics and patterns of the processed data are likely to change over time. Hence, while new clusters can be observed, other clusters need to be removed as they become irrelevant. This concept drift needs to be addressed when processing streaming data. The following methods are commonly used for handling concept drift. We show a comparison in Table 3.1.

Sliding Windows (SWs) A simple window-based approach for handling concept drift is remembering the most recent processed data points and applying outlier detection as it is done for static data. Hence, this concept can be envisioned as sliding a window over the stream of data to consider only the most recent portion of the data stream. SWs can be implemented in two flavors:

- On the one hand, we can store the n most recently seen samples in the SW with n being a constant algorithm parameter. This is the most natural approach in particular when no timestamps are associated with processed data samples or when IATs between processed samples are constant.

  If data is received with non-constant IATs, this approach has the downside that the memory length, i.e. the time span covered by the SW, is not necessarily constant and depends on how much data arrives at a certain point in time. However, the fixed number of samples that need to be stored eases implementation.

- Alternatively, we might be interested in enforcing a constant memory length, accepting that the number of samples stored in the SW might fluctuate. In particular, to provide predictable classification behavior, a memory length with a constant cut-off time in most cases is the desired behavior. Hence, all arriving

samples are associated with a time of arrival and only when the age of a sample has surpassed a certain threshold, the sample is deleted.

Depending on the scenario, this fluctuation of stored samples might be severe, which, in particular if algorithms do not have linear computation complexity in the number of SW samples, might cause substantial demands in hardware resources.

Beneath computational demands, a major difference between both approaches concerns fluctuations of the point density in feature space. It strongly depends on the investigated type of data, which of both approaches provides benefits in this respect. For instance, if the data stream is stationary with individual clusters staying active throughout all time, while only the amount of observed data samples varies over time, a SW with a fixed number of samples can capture the characteristics of the data stream well and densities within the SW do not vary substantially. In many cases, this is relevant for the functioning of outlier detection or the expressiveness of the provided outlier score.

SWs can be considered the classical way to approach concept drift, which is used by most outlier detectors devised so far. In particular, it is used by kNN outlier detectors like AbstractC [233], Exact- and Approx-STORM [32] and COD [146], but also by more modern approaches like Loda [181], RS-Hash [198] or Robust Random Cut Forest (RRCF) [108].

Reference Windows (RWs) Another frequent windowing mechanism is that of RWs, which is used in particular if the outlier detection method requires expensive building or rebuilding of a model like, e.g., with tree-based methods. Since model rebuilding needs computational effort, it is usually not possible to perform retraining after every processed data sample. Hence, for RW-based methods retraining is only performed occasionally after a fixed number of samples. Outlier scoring is thus based on the n samples that were observed before the last retraining, i.e. is based on the RW.

By only requiring to perform training occasionally, training can be allowed to be a computationally expensive procedure, which in many cases allows fast querying of the model when scoring newly observed samples. Hence, RW methods typically provide good runtime complexity. A downside is that a relatively old model is used for scoring given observed samples, i.e. very recent samples are not considered in scoring a given sample.

RWs have been used for HS-Trees [216] and xStream [161]. For these methods, they bring the benefit of avoiding having to implement means to remove outdated samples from the trained model.

Exponential Moving Averaging With SWs having performance issues and RWs being slow to adapt to very recent data, a windowing approach would be desirable that does not suffer from either of both drawbacks. An approach to achieve this goal is constructing algorithms where the model is composed of counters (e.g., counting samples that fall into certain histogram bins) and use Exponentially Weighted Moving Average

(EWMA). EWMA allows to avoid storing any samples after they have been processed. Temporal influence of individual samples is of exponential shape, so that also very recent samples are considered when processing newly observed samples.

Since the algorithm has to be constructed to use counters that allow EWMA, it is not possible to adapt any arbitrary algorithm for static data for this kind of windowing. For instance, in their construction of RS-Hash [198], Sathe and Aggarwal suggest a variant using a SW, but also outline a variant using EWMA.

Approaches for Identifying Outliers

Central to a streaming outlier detection algorithm is the question how to score outlierness of a given data sample, which provides the basic definition of outlierness. The following approaches are used as foundation of outlier detectors proposed in literature.

Distance-based Methods The most intuitive approach to detect outliers is probably a kNN approach, where the distance to nearest neighbors in feature space is used for scoring outlierness. Not only is this the simplest approach for outlier scoring, it also yields notable benefits in terms of interpretability, since the distance to neighboring samples can be broken down to differences in specific feature values, which are immediately understandable. Fundamentally, an important method to tune distance-based outlier detection is the choice of the used distance function. Indeed, deviating from the default choice of using L_1 distance or L_2 distance is reasonable in many cases. For instance, if the feature space contains nominal values, the Simple Matching Distance (SMD) or even a custom distance function is likely to be more appropriate to capture differences in these categorical values. Also in the static domain, simple kNN is a popular approach for performing outlier detection.

In the field of streaming data, various algorithms have been proposed for performing distance-based outlier detection in combination with a SW. Methods of this kind include the AbstractC [233], Exact- and Approx-STORM [32] and COD [146] algorithms. In general, discovery of nearest neighbors is a computationally challenging problem. Algorithms that have been developed therefore deviate in their way to cope with this task, but report identical results. To achieve this goal as efficiently as possible, these algorithms require both the required number of nearest neighbors and the radius threshold for being considered a nearest neighbor as algorithm parameters. Based on this threshold they do not provide a score for outlierness, but only report a binary inlier/outlier label.

In many scenarios in research, but also in application, it is desirable to obtain an outlier score instead of only a binary label. Outlier scores in such a scenario can be constructed in two ways:

- Using the number of nearest neighbors as fixed externally provided algorithm parameter, the distance of the k^{th} parameter can be interpreted as a score for outlierness.

- Alternatively, when fixing the search radius as externally provided algorithm parameter, the number of neighbors contained within this radius can be used as inverse score for outlierness.

To perform outlier scoring in these cases efficiently, a tree-based approach can be used. While most tree-based methods for discovering nearest neighbors are optimized for bulk data, the M-Tree [70] allows the updating of the already built tree structure and thus can be used in a streaming scenario where data samples continuously need to be added and removed from the SW.

Histogram-based Methods Another intuitive definition of outlierness is based on a probabilistic model of the observed data. Histograms can be built from the observed data and newly observed samples that correspond to low-probability regions are assigned a high outlierness.

Loda [181] is a well-known approach for outlier detection in data streams that follows this paradigm. Loda combines an ensemble of histogram-based detectors with the technique of random projections. Based on their obtained performance results, the authors show that this ensemble of weak detectors provides a strong detector of anomalies.

If histograms are built on the basis of univariate histograms and the feature space consists of a relatively small amount of dimensions, the result of histogram-based methods can be interpreted to a certain extent by analyzing how likely individual feature values are. However, this simple approach fails to take feature dependencies into account, which can deteriorate detection performance considerably. Unfortunately, when using random projections for improving detection accuracy as done by Loda, and beyond that when using a non-trivial ensemble of base classifiers, interpretability is lost.

Binning-based Methods Binning-based methods can be regarded a generalization of histograms, but they lose their probabilistic interpretation. In this case, the feature space is split into definite regions and the number of samples falling into a sample's region is relevant for obtaining an outlier score. In contrast to histogram-based methods, randomness is used to determine bins, resulting in different bin sizes for different dimensions and for different detectors in an ensemble. This approach yields the benefit of allowing operation at different scales.

Algorithms of this kind include HS-Trees [216], xStream [161] and RS-Hash [198]. A main distinctive characteristic between these methods is whether dimensions are treated independently for binning. Differences also concern the operation principle, i.e. whether splitting is done based on chain-based or tree-based structures.

Interestingly, structures that are used as basis for splitting in all cases are built independently of the data, which allows a-priori construction of data structures, but requires having at least a rough approximation of the value range of the used features.

Data structures used by such methods are highly abstract, are built in a randomized way, and typically use ensembles for good detection performance. Obtaining interpretability therefore is typically not viable for such methods.

Density-based Methods While not yet applied to streaming data, we also want to point out the possibility of density-based outlier detection. In particular, the LOF is a very popular approach for static data, which in combination with a windowing mechanism could also be used for streaming data. However, due to the high number of nearest-neighbor discoveries needed for outlier scoring, this approach incurs substantial computational resources that might be prohibitive for many scenarios.

Interpretability might be achieved when assuming that the obtained local density is well estimated by LOF, which, however, needs not to hold true in all cases. Since computations are rather involved, interpreting the entirety of performed computations is typically not possible.

Isolation-based Methods In static data, Isolation Forests (iForests) [156] are a popular novel approach for outlier detection. Similar to iForests, in the streaming domain Guha et al. have proposed the RRCF [107] approach for performing outlier detection.

When building a tree based on subsequent splits of the feature space, outliers are likely to be separable using just a few splits due to their anomalousness. However, at the same time they complicate the description of all remaining data, since the description must additionally include the information that contrasts them with anomalous behavior. RRCF thus is based on the idea that the description of remaining data becomes simpler when removing an outlier, i.e. the number of splits decreases. In contrast to the iForest data structure known from static data, the tree structure is optimized for steadily adding and removing points from the tree structure as it is needed for SW operation.

Similar to binning-based methods, the used tree structures typically are highly abstract, disallowing the interpretation of obtained results.

3.1.2 Comparison

We show in Table 3.2 streaming outlier detection methods that have been proposed in literature. Table 3.1 shows an overview of methods for handling concept drift and depicts their main characteristics.

Interpretability is a major research focus of this thesis. With respect to interpretability, we thus conclude that distance-based methods allow the best and most straightforward interpretability. Histogram-based methods allow interpretability only in the simplest cases, which are typically insufficient for obtaining good detection results.

Table 3.2: Outlier detection methods.

	Outlier Identification	Windowing	Ensemble-based	Uses projections	Constant space complexity	Constant time complexity	Year of publication
AbstractC [233]	Distance-based	SW					2009
STORM [32]	Distance-based	SW					2007
COD [146]	Distance-based	SW					2011
Loda [181]	Histogram-based	SW	✓	✓		˜	2016
RRCF [108]	Isolation-based	SW	✓				2016
RS-Hash [198]	Binning-based	SW or EWMA	✓	✓		✓	2016
xStream [161]	Binning-based	RW	✓	✓	✓	✓	2018
HS-Trees [216]	Binning-based	RW	✓		✓	✓	2011

3.2 dSalmon: Efficient Outlier Detection in Python

Notice of adoption from previous publications (Section 3.2)
Parts of the contents of this section have been published in the following paper:

> *[120] Alexander Hartl, Félix Iglesias, and Tanja Zseby. dSalmon: High-Speed Anomaly Detection for Evolving Multivariate Data Streams. In* 16th EAI International Conference on Performance Evaluation Methodologies and Tools. *ACM, 2023*

Software development, experimental evaluation and paper writing has been conducted by myself. Software testing and experimental evaluation was assisted by Félix Iglesias. All authors contributed in paper improvement, discussion of experiments and proofreading.

In a world of ever-increasing transmission rates, performing knowledge discovery on data streams becomes more and more challenging. Detecting anomalies and outliers being a particularly important and well-known task for data stream processing, we now turn to the problem of performing anomaly detection in data streams *efficiently*.

While algorithm implementations with low processing speed might pose a problem for practical online processing of data, it is even more challenging for researchers. Performing feature selection or fitting algorithm parameters, e.g. via grid search, involves a vast amount of algorithm runs on a captured data stream, which naturally should cover a

period that is as long as possible. Additionally, researchers often have the requirement to test algorithms on multiple datasets. Hence, the computational ability to process a data stream in real time is far from sufficient for research and slow algorithm implementations become a burden or even prevent researchers from considering specific methods for a given task.

Efficient data processing in scripting languages is mostly achieved in terms of *vectorization*, i.e. computations are performed on batches of data instead of on individual scalars to reduce the impact of interpretation overhead. Considering opportunities for vectorization, we note a basic systematic difference when processing evolving streaming data compared to the processing of static datasets. Due to the lack of an inherent ordering, for static datasets, blocks of data samples can be evaluated against one and the same model, in many cases yielding opportunities for vectorization and, hence, fast processing. The same cannot be said about evolving streaming data. Since algorithms continuously adapt to newly seen data, the model relevant for mining data sample n is potentially influenced by all data samples $0, \ldots, n-1$. Any attempt for faster processing by evaluating a block of data against the same model would therefore yield inaccurate results compared to production use, where data samples are processed one at a time at the time of their arrival.

We present our **D**ata **S**tream **A**nalysis Algorith**m**s f**o**r the Impatie**n**t (dSalmon), a framework for performing outlier detection on multivariate evolving streams of data, which has been specifically designed to process data as efficiently as possible with respect to both execution time and memory footprint. dSalmon provides a simple and intuitive Python interface to allow rapid development by data scientists, but performs processing in C++, achieving substantial performance benefits compared to existing implementations. It is easily extendable by deploying software for automatically generating boilerplate code and has almost no package dependencies.

We perform a thorough comparison of dSalmon to PySAD, the only Python framework for performing outlier detection on stream data to date. To provide a comprehensive evaluation, we measure run time, memory usage and energy consumption when applying several outlier detection algorithms on three different publicly available benchmarking datasets. Our findings show that dSalmon provides substantial benefits with respect to all measurement readings. In particular, execution time improvements by up to three orders of magnitude can be obtained with dSalmon.

The remainder of this section is structured as follows. In Section 3.2.1, we highlight related software projects to motivate our work by illustrating the gap in existing software projects our dSalmon fills. In Section 3.2.2, we then discuss design objectives, our resulting architectural design and the interface of dSalmon. To demonstrate that substantial performance benefits can be obtained with our deployed architecture, we proceed in Section 3.2.3 with a comprehensive experimental evaluation and comparison of resource consumption when using our framework.

	SW-DBOR	SW-KNN	SW-LOF	Loda [181]	RS-Hash [198]	RRCF [108]	HS-Trees [216]	xStream [161]	SDOstream [117]
Windowing mechanism	SW	SW	SW	SW	SW	SW	RW	RW	EW
Constant space complexity	×	×	×	×	×	×	✓	✓	✓
Constant time complexity	×	×	×	×	✓	×	✓	✓	✓
Ensemble-based	×	×	×	✓	✓	✓	✓	✓	×
Parallelizable in dSalmon	×	×	×	✓	✓	✓	✓	✓	×

Figure 3.1: Available methods for streaming outlier detection.

3.2.1 Related Projects

The most important software project related to dSalmon is the recent PySAD [234] framework, which similarly targets the processing of streaming data in Python. PySAD provides several methods for outlier detection on data streams and is entirely written in Python. Some of the outlier detectors PySAD provides wrap existing Python solutions, like, e.g., the PyOD [245] framework or scikit-learn [179]. In this thesis, we compare runtime performance with PySAD, since algorithms implemented in PySAD are similar to the ones we provide.

For outlier detection tasks, PyOD [245] or scikit-learn [179] can also be used directly. PyOD is a popular Python package that provides several methods for outlier detection. Unlike dSalmon, it targets methods for processing static datasets rather than streaming data. Several outlier detectors for static datasets are also provided by scikit-learn, the most popular Python framework for ML and data mining.

An important software project for processing streaming data is the MOA [51] framework implemented in Java. Algorithms provided by MOA are not limited to outlier detection, but cover several fields of data mining like clustering and classification. Several outlier detectors for streaming data are implemented in MOA, which, however, are limited to a distance-based outlier definition and provide only binary labels instead of outlier scores. In dSalmon, we additionally implement several recent approaches for stream outlier detection like ensemble-based methods. MOA is meant to be used as a stand-alone application rather than a programming library. Since implemented algorithms additionally differ severely from algorithms implemented in dSalmon, we do not include MOA in our experimental evaluations.

In the Java community, also the ELKI [201] framework is worth mentioning. ELKI provides a comprehensive selection of data mining algorithms. However, similar to PyOD, it focuses on the processing of static datasets instead of streaming data processing.

3.2.2 Architectural Design and Interface

We now describe how we engineered dSalmon to optimize usability for research on streaming data processing. To motivate our architectural design, we start by depicting software goals and resulting design decisions.

Design Decisions

We primarily target researchers working with streaming data who aim to develop or optimize systems and algorithms and therefore possess offline datasets of captured data streams. Here we present the objectives that motivated our design decisions for dSalmon.

- **High Speed Data Processing.** A primary objective is the optimization of execution time of the algorithms provided by dSalmon. This is especially important for researchers who conduct tests with different algorithms and parameters, aiming to make an informed decision about the best parameterization.
  Design Decision: In order to achieve high-speed processing, the core of dSalmon is implemented in C++. Furthermore, a spatial indexing data structure is used whenever nearest neighbor and range queries are required.

- **Straightforward Usability.** In the field of data science, Python is a commonly used language. Many tools provide Python interfaces and researchers often develop algorithms in Python. Therefore, despite being largely implemented in C++, we aimed to provide a familiar, pythonic interface.
  Design Decision: For outlier detectors, we adhere to the known interface of scikit-learn, which also has been adopted by PyOD.

- **Processing of Recorded Data.** While in the application phase data samples have to be processed one at a time, during algorithm development and parameter tuning, researchers commonly can make use of datasets consisting of previously collected streaming data. If desired, dSalmon allows providing blocks of data as input to achieve superior processing speed. In such cases, to accomplish a behavior equivalent to application phases, the implementation must guarantee to be invariant of the used block size.
  Design Decision: To achieve efficient processing and provide block size invariance, we process block samples sequentially within our fast C++ backend.

- **Support for Efficient Ensemble Learning.** In recent research, it has become common to construct data mining algorithms by pooling the results of an ensemble of weak learners, thus providing opportunities for embarrassingly parallel processing. These opportunities for parallelization have to be passed on the user, allowing a substantial speedup on modern computing hardware.
  Design Decision: We allow configurable parallelization by simply setting an `n_jobs` parameter.

- **Reproducibility.** In order to support reproducibility of results, results obtained from randomized algorithms have to be parameterized by a random seed, so that results are deterministic and reliably reproducible when providing the same seed value. Changing the used block size or the number of parallel computing threads has to leave the obtained results unaffected.
  Design Decision: We support parameterization of randomized algorithms by a random seed.

- **Simple Installation and Maintenance.** To reduce the surface for version incompatibilities and provide an uncomplicated installation, it is highly beneficial to keep the number of software dependencies small.
  Design Decision: For installing dSalmon, we only require NumPy [113]. While dSalmon uses SWIG [44] for generating Python wrapper code and makes intensive use of Boost [109], the permissive licenses of SWIG and Boost allow us to ship any code required for compilation together with dSalmon.

Besides the above goals, it is also worth elaborating on some aspects that we explicitly *do not* pursue in the development of dSalmon.

- **Algorithm Modifications.** A clear non-goal is the optimization of any of the implemented algorithms for outlier detection accuracy. Such improvements inherently depend on the specific problem under investigation and, hence, are difficult to be made in an objective way. Rather, we follow the descriptions of the respective algorithm authors as closely as possible, so that users of our framework can be confident to deploy an established, well-tested method that has typically undergone the peer review process, for their research tasks.

- **Code Redundancy.** We see the focus of our framework in filling an important gap by providing highly efficient processing for streaming data. On the other hand, for several recurring tasks of data analysis and processing, well-functioning and comprehensive tools are provided by existing frameworks like NumPy [113], scikit-learn [179] or SciPy [224], or can trivially be implemented in a fast, vectorized manner. We explicitly avoid reimplementing tools that are already provided by established software projects to keep our code base narrow and relieve the user from having to choose between competing implementations. For instance, comprehensive metrics for evaluating the quality of an obtained outlier scoring are provided by scikit-learn, like, e.g., the ROC-AUC score or the P@n score.

Architecture

Considering design decisions in the previous Section 3.2.2, it was of importance for us to allow use of dSalmon from a programming language that is known and used by data

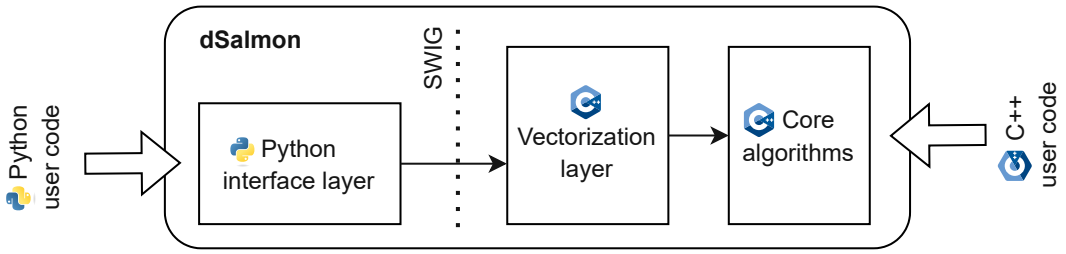


Figure 3.2: Architecture of dSalmon.

science researchers. For this reason, we targeted the Python programming language, which allows efficient and swift data science development.

Traditional data mining algorithms for static data in many cases have at least limited opportunities for vectorization. Therefore, algorithms for static data often allow efficient implementations from an interpreted language like Python directly. However, when processing a data stream the model has to be adapted for each processed point, inherently making vectorization hardly feasible, if not impossible.

To provide superior processing speed while allowing use from Python directly, dSalmon therefore implements core algorithms in C++, but provides interfaces to the algorithms from both C++ and Python.

Figure 3.2 depicts the architecture of dSalmon. Hence, the core algorithms layer depicted in Figure 3.2 is implemented in C++. We use C++ template programming for instantiating single and double precision floating-point variants of all algorithms. Researchers can thus achieve a smaller memory footprint and faster processing times by falling back to single precision processing if required. Since loop iterations are fast in C++, the core algorithms C++ interface accepts individual samples instead of blocks of data.

On the other hand, looping over individual samples in Python would incur a substantial performance penalty. Hence, the C++ vectorization layer in Figure 3.2 accepts blocks of streaming data and iterates over samples within each block when passing on the data to core algorithms. Additionally, the vectorization layer ensures that opportunities for parallel processing are efficiently taken by, for example, executing base detectors of ensemble methods in parallel.

For generating the actual interface between Python and C++, we deploy the SWIG [44] tool. The benefits of deploying SWIG are that the code base of dSalmon can easily be extended, leaving the generation of boilerplate interface code to SWIG. Since SWIG supports a wide range of target languages, our approach additionally yields the possibility to create bindings for further programming languages like R without having to rewrite core algorithms.

The Python interface layer depicted in Figure 3.2 finally accepts blocks of streaming data from user code. It accomplishes the tasks of ensuring a clean, pythonic interface and performs several sanity checks on the provided data blocks.

Listing 3.1: An example for finding the 5 most outlying points using dSalmon.

```
from dSalmon.outlier import HSTrees
import numpy as np

detector = HSTrees(window=500, n_estimators=100, n_jobs=4)
data = np.load("data.npy")
outlier_scores = detector.fit_predict(data)
outliers = np.argsort(outlier_scores)[-5:]
print("Outliers:", outliers.tolist())
```

We genuinely believe that source code should be publicly available and therefore distribute dSalmon under the LGPL 3.0 license, which permits widespread use, but requests developers to keep modified versions open-source.

Using dSalmon for Outlier Detection

Listing 3.1 shows an example of performing outlier detection with dSalmon. In this example, the rows of `data` are interpreted as sequentially arriving samples of a data stream.

As alternative to the depicted listing, a user might similarly call `fit_predict()` sequentially with blocks of consecutive rows, or even iterate over rows in `data` individually. Since data rows are iterated by dSalmon, all three approaches provide equal results. Choosing a too small block size, however, might result in substantially slower processing. As described in Section 3.2.2, block size invariance is crucial for evaluating algorithms in a realistic manner.

M-Tree Indexing

When developing algorithms for data mining, a frequent task is to find nearest neighbors in a large set of points. This requirement gave rise to the development of various indexing data structures for performing nearest neighbor and range queries efficiently. However, many indexing trees are optimized for tree construction from bulk data and do not allow removing points and inserting new points after the tree has been built. In particular, this limitation applies to the popular KDTree and BallTree data structures provided by scikit-learn [179].

dSalmon implements an M-Tree [70] spatial indexing data structure for its internal use, which allows efficient nearest neighbor and range queries in metric spaces. By using an M-Tree, dSalmon thus allows to modify the tree after it has been built.

To allow algorithm development from Python, we provide a Python interface for directly using an M-Tree in custom algorithms. Similar to our further implementations, we ensured that parallel processing capabilities can efficiently be made use of and allow

partially parallelized tree building and fully parallelized tree querying in an uncomplicated way by simply setting respective parameters.

3.2.3 Experimental Evaluation

In the following, we present results from an extensive experimental evaluation that we have performed to compare resource consumption of dSalmon with PySAD. Our algorithm benchmarks have been performed on desktop machines equipped with Intel i7-4770 processors, 16GB of main memory and no configured swap space. All machines used for evaluation have an equal setup. We use CPython version 3.7.3 and Debian Buster with kernel version 4.19.0. To avoid distorted measurements, we avoided any simultaneous use of the machines and shut down background processes as far as possible. For measuring energy consumption, we used the Running Average Power Limit (RAPL) [73] feature of our Intel CPUs, and sum memory and processor power consumption. Reported execution times do not include the time needed for loading the dataset into memory.

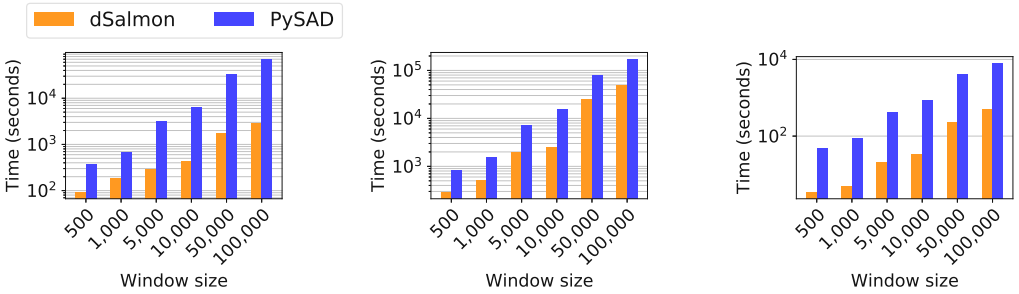
For performing realistic benchmarks, we selected publicly available datasets representing multivariate streaming data:

- The SWAN-SF [33] dataset provides measurement data on solar flares. To follow established preprocessing steps for SWAN-SF, we used preprocessing scripts available on the Internet [18], extracting the same features that repository authors used in their examples. The preprocessed dataset consists of 331,185 data samples with 12 features per sample. For assessing outlier scores, we have assigned a normal label to the majority class and marked remaining classes as outliers.

- The KDD Cup'99 [4] dataset is an established dataset for outlier detection, containing host and network based features for detecting attacks in computer networks. We marked attack samples as outliers over normal traffic and used one-hot encoding for nominal features. The resulting dataset has 4,898,431 data samples with 52 features each.

- The CIC-IDS-2017 [203] dataset, which has been introduced in Section 2.7, similarly aims at detecting network attacks, but only provides network traffic, making unsupervised attack detection substantially harder. We used an established feature vector for network traffic [230] together with publicly available preprocessing scripts [164] and considered network attacks as outliers over normal traffic. The resulting preprocessed dataset has 2,317,922 data samples and 33 features.

We selected PySAD as framework to compare it against dSalmon and performed all benchmarks using double-precision floating point processing. As described in Section 3.2.1, further software projects exist for handling outlier detection tasks. However, the majority of these projects do not provide methods for processing streaming data. Furthermore, as well as with dSalmon, PySAD can be used from Python.



(a) CIC-IDS-2017 (b) KDD Cup'99 (c) SWAN-SF

Figure 3.3: Execution time comparison for nearest-neighbors-based streaming anomaly detection.

Nearest-Neighbors Algorithms

A simple approach for establishing the outlierness of arriving data points is counting the number of nearest neighbors within a pre-determined radius. Hence, in many traditional publications [233, 32, 146] an arriving data point is declared to be an outlier if less than $k \in \mathbb{N}$ point of the current SW lie within a radius $R \in \mathbb{R}^+$. While modern approaches for anomaly detection frequently outperform this simple nearest-neighbors-based approach in both, detection accuracy and execution time, the importance of a simple nearest-neighbors-based approach lies in its unrivaled interpretability of provided outlier scores, making its availability crucial for dSalmon.

While providing a binary label in some cases is sufficient in practice, for research and parameter selection it is usually necessary to obtain a score for outlierness for data samples. As described in Section 3.1, when requiring an outlier score instead of a binary label, distance-based outlier detection can be performed in two flavors:

1. When implementing a SW-based kNN rule, outlier detection can be parameterized by the neighbor count k, providing the distance to k^{th} nearest point as outlier score.

2. Alternatively, outlier detection can be parameterized by the search radius R, providing the number of neighbors within R as inverse outlier score.

dSalmon allows outlier detection using both flavors (1) and (2), termed SW-KNN and SW-DBOR, respectively, and deploys M-Tree [70] indexing to reduce execution time.

Figure 3.3 shows the execution times of the ExactStorm model of PySAD, which similarly implements nearest-neighbors-based outlier detection, and the SW-DBOR model of dSalmon for different lengths of the SW. Since PySAD provides nearest-neighbors-based outlier detection only in flavor (2), we use this mode of operation also in dSalmon for the comparison. To provide a meaningful comparison, for each individual window size we used grid search on a logarithmic scale for finding the radius R that optimizes the

ROC-AUC score when applying the algorithm to the complete dataset, and used the resulting R for performing the benchmark.

Execution time benefits demonstrated in Figure 3.3 can be explained by two effects: On the one hand, for small window lengths execution time is dominated by interpretation overhead of the Python language for PySAD, which can be avoided by dSalmon due to its C++ core implementation. However, PySAD performs distance computations for each processed data sample in a vectorized manner, diminishing the interpretation overhead as the window length increases.

As shown in Figure 3.3, dSalmon is able to retain a substantial speedup even as window size increases. This observation demonstrates execution time benefits of M-Tree indexing compared to straight per-sample distance computations.

Ensemble-based Outlier Detectors

In recent research, an increasingly popular approach for outlier detection, which sets new records in detection accuracy, is to construct algorithms by averaging outlier scores obtained by an ensemble of weak learners. Beneath yielding good accuracy, this approach is intrinsically embarrassingly parallel, as the processing of distinct base detectors can trivially be distributed to several workers. dSalmon makes it easy to leverage this feature by simply setting an `n_jobs` parameter.

When evaluating execution performance, for the sake of providing a fair comparison, we chose algorithms whose specification leaves little room for interpretation. In particular, we selected the following methods:

- RRCF [108] uses an ensemble of dynamically constructed trees, where each tree performs random cuts based on the feature space of observed samples. Concept drift is taken care of using a SW approach. We perform runs with varying window sizes to show dependence on this parameter.
- Half-Space-Trees [216] similarly constructs an ensemble of trees, but performs tree construction statically at the time of algorithm initialization. Concept drift is considered based on a RW approach. In our experiments, we vary the depth of the constructed trees to evaluate influence of tree depth on resource usage. Since PySAD does not support setting the `sizeLimit` parameter described in [216], we similarly set `sizeLimit=0` for dSalmon.
- xStream [161] is a recent method for outlier detection, which introduces half-space-chains, which find an anomaly score by splitting randomly selected features with varying precisions. xStream combines half-space-chains with the technique of

[1] Missing results for RRCF using PySAD indicate experiment runs that failed due to reaching Python's recursion limit.

[2] xStream for KDD Cup'99 using PySAD with 50 random projections failed due to running out of memory.

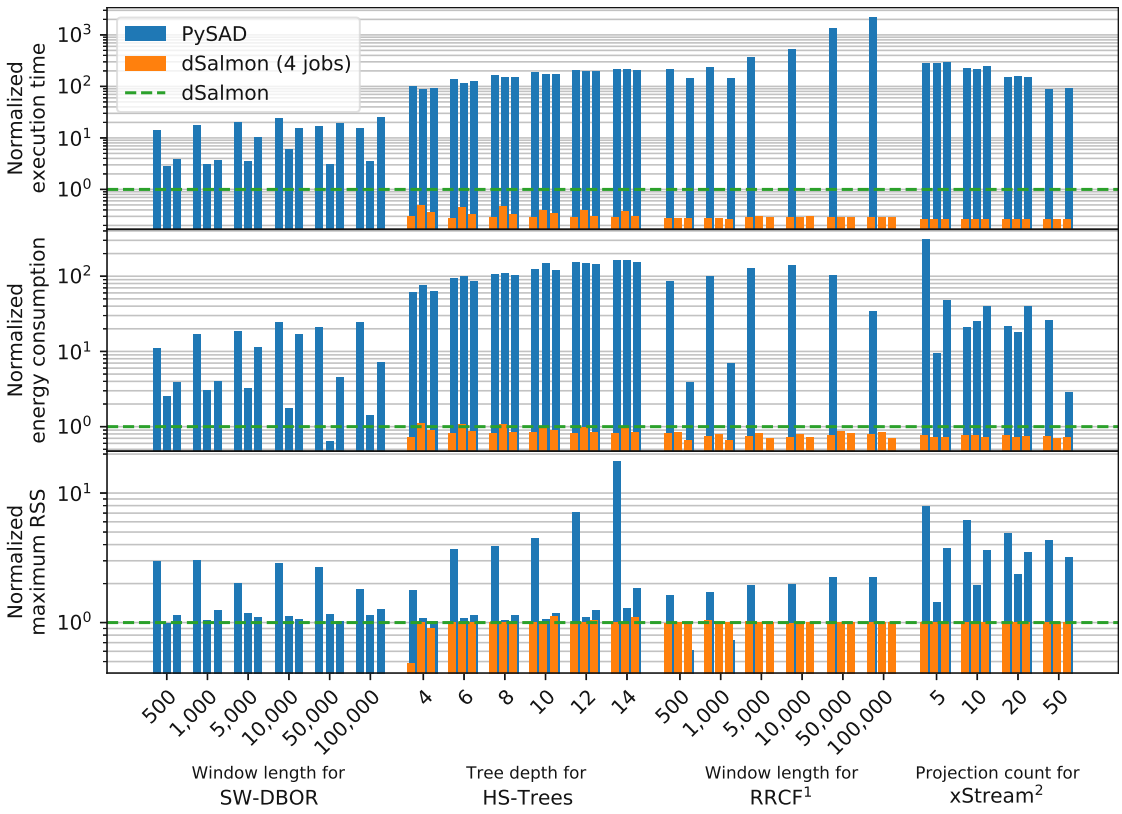
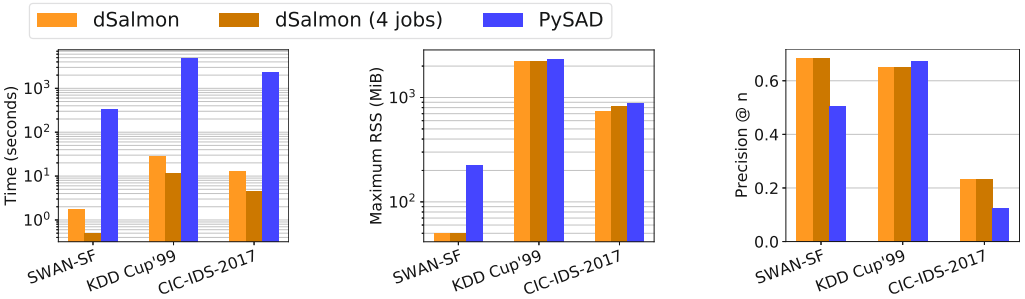


Figure 3.4: Overall comparison of the resource consumption of several outlier detectors implemented by dSalmon and PySAD. For each parameter setting, values are normalized to results obtained by single-threaded dSalmon for better comparison. Bars depicted for each algorithm parameterization indicate results for SWAN-SF, KDD Cup'99 and CIC-IDS-2017 in this order.

random projections. For benchmarking, we set the chain length to 15 as used for the evaluations in [161] and vary the number of projections to show dependence on this parameter.

In our experiments, we use an ensemble size of 100 for all algorithms, which is similarly used as default value for ensemble methods by scikit-learn. Using 100 base estimators as ensemble size is a natural choice and is likely to reduce statistical variation of outlier scores to an acceptable level. For dSalmon, we evaluate performance for both single-threaded operation and when utilizing four processor cores. PySAD does not support multi-threaded operation.

In Figure 3.4 we depict a summarized comparison of resource consumption that we measured while performing outlier detection with dSalmon and PySAD on the SWAN-SF, KDD Cup'99 and CIC-IDS-2017 datasets, also including the results already presented in Section 3.2.3. We depict execution time, memory usage as maximum Resident Set



(a) Time (b) Memory (c) P@n

Figure 3.5: Time, memory and outlier detection performance for HS-Trees using a tree depth of 10.

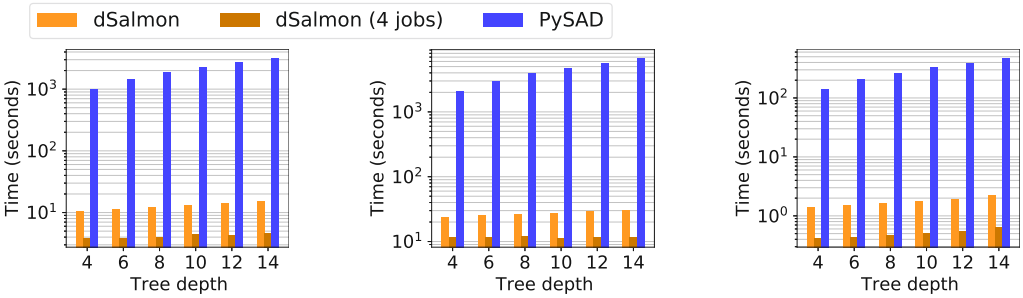
Size (RSS) and energy consumption. Since we aim to depict relative performance when using both frameworks, we normalize all measurements by results obtained when using dSalmon on one processor core.

Figure 3.4 shows that, particularly for modern ensemble-based algorithms, dSalmon yields substantial execution time benefits compared to PySAD. For most algorithm runs, a speed-up by a factor of more than 100 can be obtained. From Figure 3.4 we can additionally conclude that dSalmon makes highly efficient use of parallel processing capabilities. Execution time indicates that, by using four simultaneous jobs, a speed-up of almost 4 can be obtained in most cases. Furthermore, memory consumption does increase when relying on parallel processing, allowing highly efficient operation on modern multi-core desktop machines or servers. It is also interesting to note that by relying on multi-threaded processing considerable energy savings can be obtained.

In what follows, we will analyze behavior for specific algorithms in more detail. We skip RRCF for closer analysis, as for RRCF many PySAD runs failed due to reaching the maximum recursion limit. This error cannot be fixed trivially, since increasing CPython's maximum recursion limit can result in overflowing the program stack [232]. Furthermore, since algorithm runs performed with SWAN-SF indicate a severe dependence of window size, we find the RRCF implementation adopted by PySAD generally not suitable for analyzing datasets of large size.

Half-Space-Trees Figure 3.5 shows the absolute measurement readings that we obtained for HS-Trees with a tree depth of 10. In the light of execution times in Figure 3.5 (a), we can generally attest HS-Trees' outstanding run time performances, since HS-Trees is able to process millions of data samples in about 10 seconds.

It is worth noting that, besides obtaining markedly different execution times, we also obtained deviating results for detection performance when deploying dSalmon and PySAD, as shown in Figure 3.5 (c). Since outlier scores are averaged from 100 base detectors, it is unlikely that these differences are the sole result of inherent algorithm randomness.



(a) CIC-IDS-2017 (b) KDD Cup'99 (c) SWAN-SF

Figure 3.6: Runtimes of HS-Trees in response to variations of the tree depth.

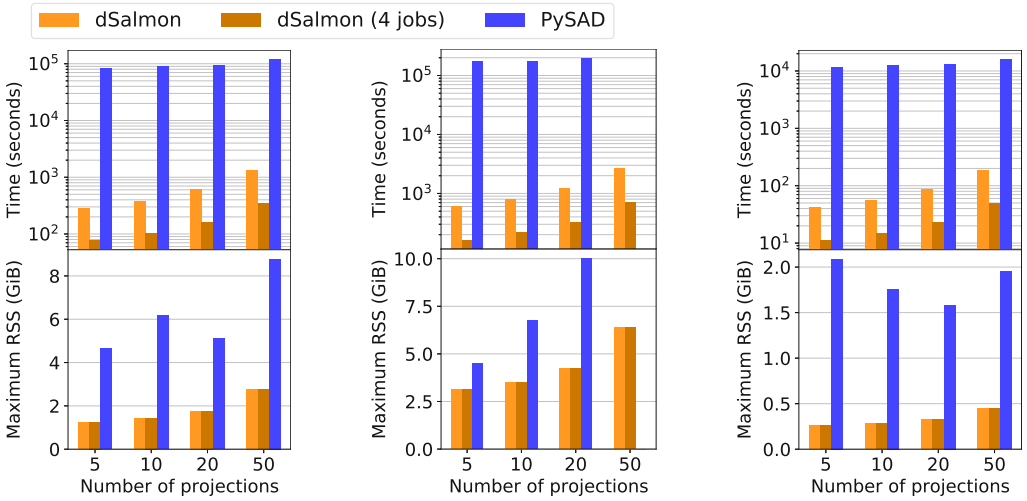
Attempting to find the reason for this deviation, we analyzed implementation details of HS-Trees in PySAD and noticed deviations in the computation of outlier scores. In particular, outlier scoring in PySAD's HS-Trees deviates slightly from the method described in [216], as it bases the reported outlier score on all traversed nodes instead of just the terminal node. dSalmon, on the other hand, computes outlier scores only from terminal nodes as described in the original publication introducing HS-Trees [216].

Figure 3.5 (c) additionally demonstrates reproducibility of obtained results for dSalmon. Hence, for both independent algorithm runs – one utilizing one processor core and another utilizing four cores – the precise same outlier scores are reported, since the same seed value has been provided as parameter. This holds true even though both runs differ in their parameterization for parallel processing. We have verified this property also for all further runs depicted in Figure 3.4.

In Figure 3.6, we depict absolute execution times as function of tree depth. Hence, dSalmon is approximately 100 times faster than PySAD and execution time shows only a slight increase when increasing the tree depth. In fact, rather memory occupation is limiting the usable maximum tree depth, since for HS-Trees tree structures are statically created, resulting in memory consumption that increases exponentially with tree depth.

xStream Figure 3.7 shows absolute execution times und maximum RSS as function of the number of projections for xStream. As discussed by the algorithm authors [161], execution time depends linearly on ensemble size, chain length and the number of projections. Hence, observed behavior for our dSalmon runs is reasonable. We notice that PySAD shows a less pronounced dependency of the number of projections. Consequently, execution time differences of dSalmon range between 20 and 200. dSalmon proves to make highly efficient use of parallel processing, allowing a 4-times speed-up by using 4 parallel jobs.

For the PySAD implementation, dependence of memory consumption as a function of the number of projections, as depicted in Figure 3.7, is particularly striking. While for SWAN-SF memory consumption shows no clear dependence of the number of projections,



(a) CIC-IDS-2017 (b) KDD Cup'99 (c) SWAN-SF

Figure 3.7: Runtimes of the xStream algorithm in response to variations of the number of projections.

for KDD Cup'99 a clear monotonic dependence can be observed. For 50 projections, xStream eventually fails on our 16GB machine due to running out of memory.

Analyzing implementation details in PySAD and comparing to dSalmon's implementation, we noticed that both implementations differ in the technique for counting bin frequencies. Algorithms authors [161] suggest using a Count-min sketch (CMS) [71] for this purpose to ensure constant space complexity. In dSalmon, we adopt the approach of using CMSs, while PySAD uses classical hash tables. The use of hash tables for this purpose provides an explanation for the data-dependent memory consumption observable in Figure 3.7, since memory consumption in this case is reduced if the majority of data samples shares a small number of bins.

In dSalmon, memory requirements for storing CMS structures are independent of the number of projections. The increase of memory consumption can thus be explained by the memory requirements for storing projected values for a given block of data.

3.2.4 Discussion

We have introduced and presented dSalmon, a highly efficient framework for outlier detection on multivariate evolving data streams. Due to the nature of streaming data, data samples frequently accumulate to a substantial volume within short time, making efficient processing crucial. We have presented dSalmon's architecture, which allows easy extension and enables researchers and practitioners to add algorithms for outlier detection of even implementing entirely different methods for analyzing streaming data.

In a thorough evaluation, we have shown that dSalmon was able to outperform existing

Python stream outlier detectors by up to three orders of magnitude with respect to execution time. Combined with the selection of a recent outlier detection method optimized for processing high-rate data streams, gigabytes of data can be processed in few seconds, paving the way for analyzing comprehensive datasets, which increasingly become available due to advances of modern technology.

3.3 Applicability of Outlier Detection Methods to Attack Detection

Notice of adoption from previous publications (Section 3.3)
Parts of the contents of this section have been published in the following paper:

> *[131] Félix Iglesias, Alexander Hartl, Tanja Zseby, and Arthur Zimek. Are network attacks outliers? a study of space representations and unsupervised algorithms. In* Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 159–175. Springer, 2019*

Experiment execution has been conducted by myself, experiment design was joint work with Félix Iglesias. Text writing has mainly been performed by Félix Iglesias. All authors contributed in discussion of experiments and in proofreading of the manuscript.

After having had a general look at outlier detection for streaming data, we now turn to the problem of performing NID using outlier detection methods. The study of previous research on network security analysis [86] discloses three main claimed goals: (a) attack detection, (b) anomaly detection, and (c) traffic classification. These three topics are not the same, but undoubtedly overlap. For instance, traffic classifications often include classes that are attacks. An anomaly might be an attack, but an attack does not necessarily show itself to be an anomaly. The traffic features selected for the analysis obviously play a determining role to see if a network attack expresses itself as an anomaly or not, but also the analysis perspective is relevant [200]. For example, Distributed Denial-of-Service (DDoS) attacks usually appear as anomalous peaks in network monitors that observe traffic as time series [83]. However, they are hardly anomalies from a spatial perspective, in which they can take a significant portion of the total captured traffic. DDoS attacks try to harm targets by bombarding them with false connection requests. Actually, DDoS and other types of illegitimate traffic (e.g., scanning activities) have become so common that they can rarely be considered anomalies in most networks anymore [129, 76].

When the term "outlier" comes into play, things become even more confusing. "Anomaly" and "outlier" are not smooth synonyms, and even the description of *outlier* can be ambiguous in practical implementations [246]. For instance, it is common to find small groups of close traffic instances that are distant from the data bulk. Together, they form an *outlying cluster*. Considered individually, instances can be deemed as outliers or not. Even in spite of such ambiguities, in related research the meaning of *anomaly* is commonly assumed without discussion. Carefully reviewing such works in the field of IDSs (and excluding time series analysis), the empirical meaning of anomaly inferred from experiments habitually corresponds to *network attacks that show outlierness*. Some authors identify attacks as anomalies and perform their detection with outlier-based

techniques [50, 100, 242]. In addition, many outlier-based detection proposals appear in other field surveys [49, 59, 140].

Nevertheless, do network attacks actually show themselves as outliers or outlying clusters? This is the crucial point that will make unsupervised methods effective for attack detection or not. Related works frequently take it for granted, but the question must be analytically answered, considering that most attacks are designed to pass unperceived. As a starting point, we highly recommend that research works on *anomaly detection* in NTA clearly establish their definition of *anomaly*. Otherwise, whenever theoretical proposals are implemented into real scenarios – far from laboratory conditions – such methods are prone to trigger detection alarms in view of many harmless, meaningless, noisy instances. This discussion is critical because precisely unacceptable high false positives rates is what slows down the adoption of ML in real-world network attack detectors [95, 99]. If this is true for supervised ML, it is even more severe for unsupervised methods, which are also commonly evaluated with the same IDS datasets (e.g., [50, 100, 242]). Note that IDS datasets are usually not designed to match realistic ratios between normal and attack traffic, but to offer a variety of attack classes with sufficient representation in the dataset [96]. This is not ideal for unsupervised methods because they work by learning from the sample placement and space geometries drawn by the analyzed data. From here, and without considering irrelevant, easy-to-detect, illegitimate traffic that has become common, it naturally follows that the real-world ratio attack/non-attack is going to be considerably lower than in IDS datasets. Therefore, the probabilities for a detected anomaly to be an actual attack drop dramatically. How the base-rate fallacy affects IDSs was already advised by Axelsson in [38].

The previous observations do not imply that unsupervised methods are not valid for attack detection. Instead, they introduce the necessity for evaluating the outlierness of network attacks and for investigating if unsupervised methods suffice by themselves for the actual detection in real implementations. Signature-based detection or supervised approaches are limited in detecting novel threats and zero-day attacks. Therefore, the contribution of unsupervised approaches is deemed highly valuable. As already discussed above, a last challenge that unsupervised methods must additionally face is their traditional high computational complexity. Most popular outlier detection algorithms are naturally *instance-based* and show considerable time and memory demands [180, 63]. Network traffic analysis for attack detection must be fast and lightweight, since it must deal with ever-growing volumes of traffic (big data, streaming data), and is expected to react promptly when malicious instances are discovered.

The main contribution of this section is answering the following questions:

- **Are network attacks outliers?** We study five popular and recent space representations used in IDS applications and experiment with five popular and recent unsupervised outlier detection algorithms in order to elucidate if network traffic attacks show a distinguishable outlier nature.

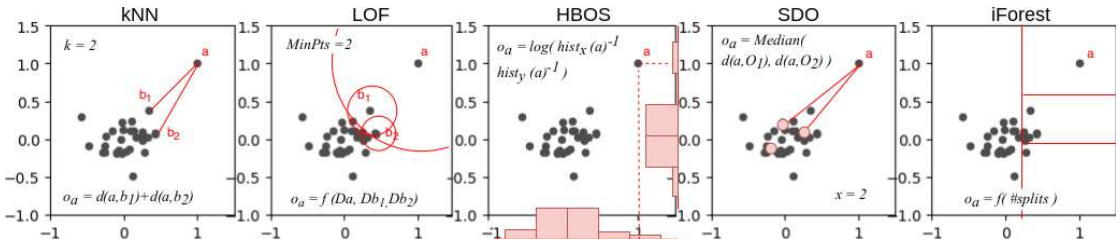


Figure 3.8: A quick overview of how the studied algorithms estimate the outlierness (o_a) of a random point a.

- **What are the most suitable feature representations for attack detection?** We investigate which existing feature vectors perform best in conjunction with outlier detection.
- **Is the observed outlierness sufficient as indicator for implementing real-world attack detection?** We discuss if the detected outlierness suffices for implementing effective detectors in real environments. Additionally, we propose a new vector that maximizes attack/non-attack separation.

We note that in this section we perform our evaluations with established outlier detectors for static data instead of streaming data. This decision is motivated on the one hand by practical limitations of available datasets, which not only in most cases cover only very short durations, hence, the temporal sequence of benign and attack traffic does not represent realistic application scenarios. On the other hand, we are also interested in which outlier definitions conform to attack traffic. The window mechanism, however, which becomes relevant in practice, is orthogonal to the basic outlier definition and can be designed to meet the demands of a particular application.

Our experiments are conducted on the CIC-IDS-2017 dataset [203] we introduced in Section 2.7, which is one of the most complete, reliable IDS evaluation datasets to date. As for the selected features, we study five vector spaces created by the CAIA [230], CISCO-Joy [31], Consensus [85], TA [129] and AGM [130] formats. Outlierness ranks are obtained by using five different algorithms: kNN [185], LOF [58], HBOS [101], iForest [155], and Sparse Data Observers (SDO) [133].

3.3.1 Outlier Detection Algorithms

In this section, we briefly introduce the used outlier detection algorithms. We have already introduced some of the principles outlined here in Section 3.1. However, in this section we put emphasis on concrete outlier detection methods as they are used for static datasets. A visual overview of the different approaches is shown in Figure 3.8.

k Nearest Neighbors (kNN). The kNN distance has been used for measuring object isolation in [185], where each instance outlierness is ranked based on the distance to its

k^{th} nearest neighbor. kNN is an instance-based method where estimations are locally approximated. It does not require training and the computational effort appears every time that a new instance must be evaluated and compared with the previous ones. kNN requires setting a k parameter.

LOF. The Local Outlier Factor (LOF) algorithm entailed a considerable enhancement in the task of measuring instance outlierness within data [58], generating a varied family of derived algorithms [200]. LOF compares the density estimate based on the k-nearest neighbors with the density estimates for each of the k-nearest neighbors, thus adapting to different local densities. LOF is also an instance-based method and does not require training. In a recent comparison, LOF has shown to be a good benchmark solution, which, in general, has not been significantly outperformed by more recent methods in terms of accuracy [63]. LOF uses the *MinPts* parameter, which is equivalent to k in kNN.

HBOS. Histogram-based Outlier Score (HBOS) [101] is a simple, straightforward algorithm based on evaluating the feature empirical distributions of the analyzed dataset (i.e., histograms for continuous features and frequency tables for nominal features). Since it assumes feature independence, it sacrifices precision to achieve fast performances in linear times. Outlierness is calculated based on the relative position of the instance feature values with regard to the obtained empirical distributions. HBOS does not require parameterization, but for the histogram binning, which allows static bin-widths (k equal width bins) or dynamic bin-widths (in every bin falls N/k instances, being N the total number of instances). In our experiments throughout Section 3.3, bins-widths are static.

iForest. Isolation Forest (iForest) [155] is a model-based outlier-ranking method that shows linear time complexity with low memory requirements even in front of large datasets. The operation principle is as follows: For a given instance, features and splits are randomly selected in a procedure that progressively reduces the range of feature values until the instance is isolated (i.e., the only instance in the remaining subspace). The number of splits defines the outlierness value of the instance, since outliers are expected to be easier to isolate (less splits) than inliers (more splits). The partitioning procedure can be abstracted as a tree (an iTree), therefore an iForest provides the weighted evaluations of a set of iTrees. During training, iTrees are built using the training dataset; in application phases, instances pass through iTrees to obtain outlierness scores. iForest parameters are: t, the number of estimators or iTrees; ψ, the sample size to train every iTree; and f, number of features passed to each iTree.

SDO. The SDO algorithm is a model-based unsupervised outlier-ranking method that has been designed to provide fast evaluations and to be embedded in autonomous frameworks [133]. SDO is conceived to avoid the common bottleneck problems implied by traditional instance-based outlier detection when a continuous evaluation of incoming instances is demanded. SDO creates a low-density data model by sampling a training population. During training, model instances – called *observers* – are evaluated in a way that low-active observers are removed. Thus, the low-density model becomes free of potential outliers. In application phases, observers provide instance outlierness based on joined distance estimations. SDO is lightweight, easy to tune, and makes the most of

pre-knowledge. SDO parameters are intuitive and stable, *rule of thumb* parameterization works well in most applications. Parameters are: k, the number of observers; x, the number of closest observers that evaluate every instance; and q (or q_v), which establishes the threshold for the removal of low-active observers.

3.3.2 Data and Evaluation of Results

We used the CIC-IDS-2017 dataset as described in Section 2.7. The CIC-IDS-2017 dataset is recent, meets several quality criteria and contains a range of different attack types, providing ideal conditions for our undertaken task in this section.

To base our evaluations on known and commonly used flow representations, we used the CAIA, Consensus, TA, and AGM vectors as outlined in Section 2.2.2. Table 3.3 summarizes feature vectors in the format used here. We apply the nomenclature described in [166]. We refer the interested reader to the original papers for detailed descriptions.

For evaluating algorithms, we have used the same metrics applied by Campos et al. in their recent outlier detection algorithm comparison [63], which we summarized in Section 2.2.3. We additionally provide the maximum F1 score [182] as additional metric to handle dataset imbalance.

3.3.3 Experiments

This section describes the conducted experiments. Henceforth, we refer to the feature formats as the subset F and the used algorithms as the subset A:

$$F = \{\text{CAIA, Consensus, TA, Cisco-Joy, AGM}\} \tag{3.1}$$

$$A = \{k\text{NN, LOF, HBOS, iForest, SDO}\} \tag{3.2}$$

We describe the experiments step-by-step:

1. Flow extraction From CIC-IDS-2017 network traces, we extracted features to match the studied representations. Therefore, for each vector format we obtained a structured dataset D_i with $i \in F$, containing the respective features plus a binary label (attack, non-attack) and a multiclass label (attack family). Feature vectors were extracted with the go-flows [227] feature extractor.

2. Cleaning and normalization We removed nominal features from preprocessed datasets (see Table 3.3), except for the "Protocol", which was transformed into the dummies "TCP", "UDP", "ICMP" and "others". Datasets were min-max normalized:

$$Z_i = \text{normalize}(\text{remove\_nominal}(D_i)) \tag{3.3}$$

Table 3.3: Studied NTA representations (feature vectors).

AGM	**Object:** Source hosts (unidirectional)			
	Key: srcIP; **Obs.window:** 10sec			
	Features (22 total):			
	#dstIP	#dstPort	#TTL	#pktLength
	mode_dstIP[1]	mode_dstPort	mode_TTL	mode_pktLength
	pkts_mode_dstIP	pkts_mode_dstPort	pkts_mode_TTL	pkts_mode_pktLength
	#srcPort	#protocol	#TCPflag	pkts
	mode_srcPort	mode_protocol	mode_TCPflag[1]	
	pkts_mode_srcPort	pkts_mode_protocol	pkts_mode_TCPflag	
Time-Activity	**Object:** Flows (unidirectional)			
	Key: srcIP, dstIP, protocol; **Obs.window/timeout:** 60sec			
	Features (13 total)			
	srcPort	maxton	seconds-active	
	dstPort	minton	bytes_per_seconds-active	
	protocol	maxtoff	pkts_per_seconds-active	
	bytes	mintoff		
	pkts	interval		
CAIA	**Object:** Flows (bidirectional)			
	Key: srcIP, dstIP, srcPort, dstPort, protocol; **Idle/active timeout:** 300sec/1800sec			
	Features (30 total):			
	protocol	max_srcPktLength	max_srcPktIAT	#srcTCPflag:fin
	duration	stdev_srcPktLength	stdev_srcPktIAT	#srcTCPflag:cwr
	srcPkts	min_dstPktLength	min_dstPktIAT	#dstTCPflag:syn
	srcBytes	mean_dstPktLength	mean_dstPktIAT	#dstTCPflag:ack
	dstPkts	max_dstPktLength	max_dstPktIAT	#dstTCPflag:fin
	dstBytes	stdev_dstPktLength	stdev_dstPktIAT	#dstTCPflag:cwr
	min_srcPktLength	min_srcPktIAT	#srcTCPflag:syn	
	mean_srcPktLength	mean_srcPktIAT	#srcTCPflag:ack	
Consensus	**Object:** Flows (bidirectional)			
	Key: srcIP, dstIP, srcPort, dstPort, protocol; **Idle/active timeout:** 300sec/1800sec			
	Features (20 total):			
	srcBytes	dstPort	mode_dstPktLength	median_srcPktLength
	srcPkts	protocol	min_srcPktLength	median_dstPktLength
	dstBytes	duration	median_srcPktIAT	min_dstPktLength
	dstPkts	max_srcPktLength	variance_srcPktIAT	median_dstPktIAT
	srcPort	mode_srcPktLength	max_dstPktLength	variance_dstPktIAT
Cisco[2]	**Object:** Flows (bidirectional)			
	Key: srcIP, dstIP, srcPort, dstPort, protocol; **Idle/active timeout:** 300sec/1800sec			
	Features (650 total):			
	srcPort		#certificates	
	dstPort		#SAN	
	packet length sequence (100 features)		offered cipherSuites (139 features)	
	IAT sequence (100 features)		selected cipherSuites (26 features)	
	byte distribution (256 features)		offered TLSExtensions (13 features)	
	public key length		accepted TLS extensions (11)	

1: Removed from the analysis.
2: The Cisco-Joy tool can extract more features. We removed features that did not contain usable information in the CIC-IDS-2017 dataset.

3. Stratified sampling Datasets were sampled and a 5% subset was drawn for hyperparameter search and tuning: $Z_i' = \text{strat\_sample}_{0.05}(Z_i)$, where $i \in F$. The sampling process was stratified with respect to the multiclass labels to keep balanced distributions.

Table 3.4: Used parameters in the experiments.

	Consensus	CAIA	AGM	TA	Cisco-Joy
kNN	$k = 2$	$k = 2$	$k = 15$	$k = 3$	$k = 15$
LOF	$MinPts = 5$	$MinPts = 5$	$MinPts = 18$	$MinPts = 5$	$MinPts = 39$
HBOS	$k = 20$	$k = 22$	$k = 992$	$k = 21$	$k = 20$
iForest	$t = 50, f = 37$ $\psi = 860$	$t = 95, f = 26$ $\psi = 873$	$t = 96, f = 2$ $\psi = 696$	$t = 64, f = 1$ $\psi = 529$	$t = 73, f = 428$ $\psi = 281$
SDO	$k = 553, x = 9$ $q_v = 0.2$	$k = 396, x = 5$ $q_v = 0.25$	$k = 823, x = 11$ $q_v = 0.2$	$k = 926, x = 23$ $q_v = 0.5$	$k = 281, x = 11$ $q_v = 0.2$

4. Hyperparameter search For each vector format $i \in F$ and algorithm $j \in A$, hyperparameter search was conducted by means of evolutionary algorithms[1]. Obtained hyperparameters are shown in Table 3.4.

5. Univariate analysis of outlierness ranks We split each Z_i dataset into a *non-attack* subset $Z_{i,n}$ and an *attack* subset $Z_{i,a}$. Later, measures of central tendency and histograms over Z_{in} and $Z_{i,a}$ were extracted with each algorithm j.

6. Analysis with outlier-ranking metrics For each dataset Z_i, the performance of each algorithm j was evaluated with the metrics defined in Section 2.2.3.

7. Feature selection for maximizing outlierness Finally, CAIA, Consensus and AGM formats (i.e., the best ones in previous experiments) were joined, vectors were extracted from PCAPs, and a 5% stratified sample was drawn, obtaining the final Z'_F dataset. By means of a forward wrapper with SDO as nested algorithm, features were gradually selected to find a set that maximizes the separation between attack and non-attack outlierness. ROC-AUC was selected as optimization criterion. The obtained vector was named "OptOut" (from Optimized Outlierness), it is shown in Table 3.6. Steps 4, 5 and 6 were repeated for the OptOut vector.

3.3.4 Results and Discussion

We proceed to show results and discuss the questions raised above.

Are Network Attacks Outliers?

Figure 3.9 shows box plots obtained from the *univariate analysis of outlierness ranks* step. For the sake of visibility, extreme values (top outliers) have been removed and outlierness ranks have been normalized. Upper and lower box boundaries correspond to 75^{th} and 25^{th} percentiles respectively, whereas upper and lower whiskers correspond to 95^{th} and 5^{th} percentiles. Additionally, we show some histograms in Figure 3.10 (attack and non-attack empirical densities are equalized by normalizing histograms). There are four immediate evidences that stand out from the statistics:

[1] https://github.com/rsteca/sklearn-deap

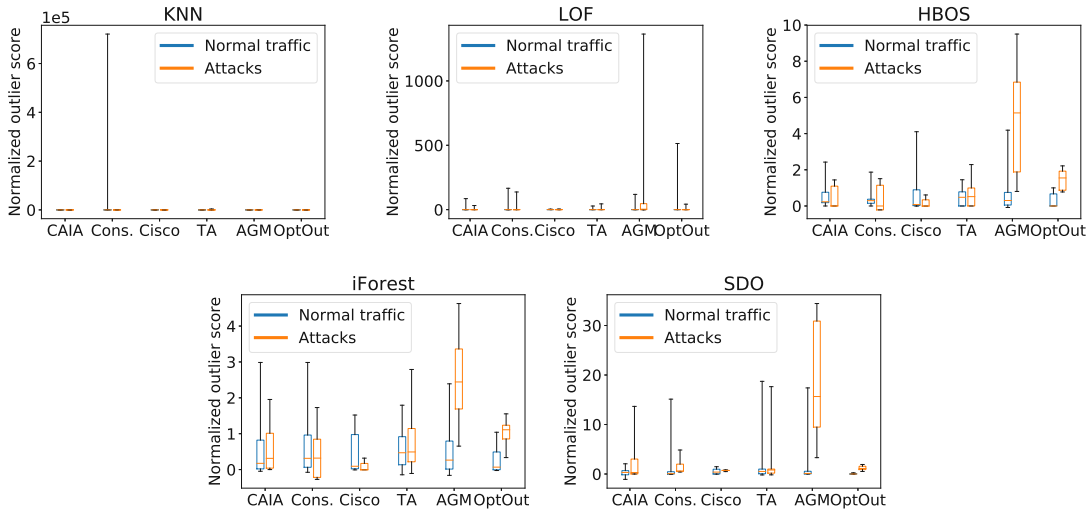


Figure 3.9: Box plots for outlierness ranks.

a) The differences between attack and non-attack instances in terms of outlierness for the Cisco-Joy are useless for discriminating attacks. Note that box plots and distributions overlap or non-attacks show higher values.

b) Regardless of the used algorithm, as a general rule attacks show higher outlierness than non-attack instances when using the CAIA, TA or AGM vectors, being AGM the format that shows major differences.

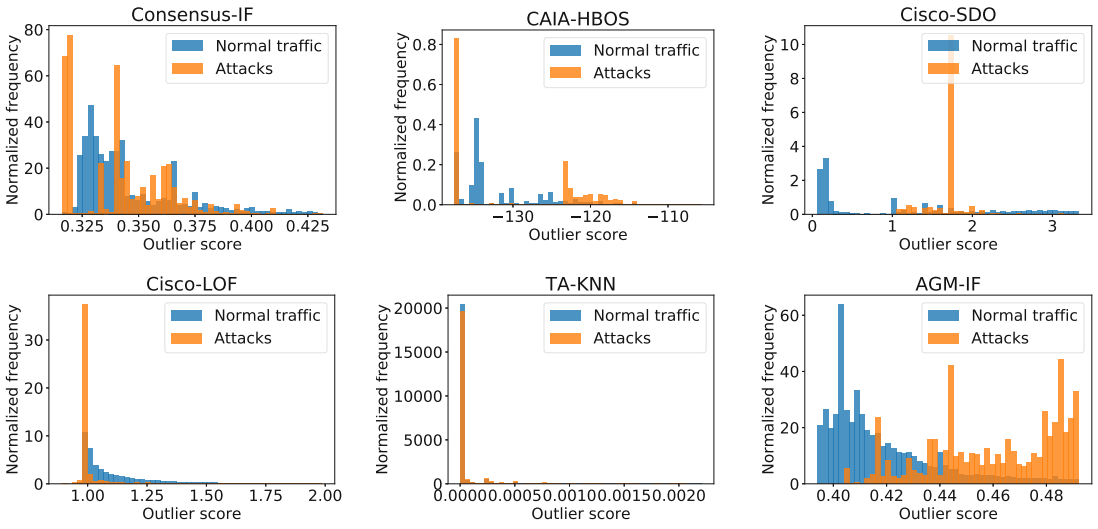


Figure 3.10: Normalized histograms (top 5% outliers removed for a better visualization).

c) Attack and non-attack outlierness ranges significantly overlap.

d) SDO shows the best performances, followed by HBOS.

The inability of the Cisco-Joy format for discriminating attacks based on outlierness (a) was expected since this vector uses a considerably high dimensional space with a majority of binary features, drawing an input space highly unsuitable for methods based on Euclidean metrics. On the other hand, the preponderance of SDO and HBOS (d), when considered together with observations (b) and (c), suggests that network attacks tend to be global, but clustered outliers, and not local outliers. The spaces drawn by the feature vectors are highly noisy and rich in density variations, and such noise and multiple densities are mainly generated by legitimate traffic. Network attacks tend to set small clusters relatively far from the data bulk. Such conditions favor non-local distance-based methods like HBOS and SDO. In any case, the significant range overlap (c) makes detection solely based on outlier-ranking algorithms hardly suitable for real applications, in which high false positive rates would be unacceptable.

What Are the Best Feature Vectors for the Task?

Table 3.5 shows the performance of algorithms for each feature vector with the indices defined in Section 2.2.3. As for the algorithms, the evaluation measures corroborate the findings discussed in Section 3.3.4, confirming the prevalence of HBOS and SDO. On the other hand, noteworthy is the fact that the AGM vector shows high ROC-AUC and low values for other indices, whereas CAIA and Consensus show low ROC-AUC but higher values for the other indices when compared with AGM. This fact suggests that, in the AGM case, most attacks show higher outlierness than most non-attack instances, but still top outlierness values correspond to legitimate traffic. Contrarily, in the CAIA and Consensus cases, most attacks and most non-attacks show similar outlierness, but top outlier positions are considerably taken by attacks (note that attacks in the dataset are negligible compared to normal instances). Such circumstance favors the use of the AGM vector to build a general-purpose detector, but CAIA or Consensus as a support detector for evaluating only extreme outlierness cases. More interestingly, it suggests that vector formats are complementary and a new feature vector that maximizes attack outlierness can be built from them.

Can We Improve Vectors and Use Them in Real Detection?

Results in Table 3.5 show that the studied vectors would generate many false positives in real-world applications. As described in Section 3.3.3, we constructed a feature vector OptOut that maximizes the separation between attack and non-attack outlierness. OptOut uses the 5-tuple key, but enriched with features that describe the behavior of the network device as information source, therefore uses application-based and endpoint-based behavior at the same time. Table 3.6 shows the included features in the OptOut vector and Figure 3.11 the forward selection process. We performed hyperparameter search

Table 3.5: Algorithm performances.

		P@n	AP@n	AP	AAP	Max. F1	ROC-AUC
Consensus	HBOS	0.40	0.20	0.26	0.01	0.44	0.42
	LOF	0.22	-0.04	0.20	-0.07	0.41	0.47
	kNN	0.18	-0.10	0.06	-0.26	0.41	0.47
	iForest	0.20	-0.07	0.12	-0.18	0.41	0.40
	SDO	0.58	0.44	0.40	0.20	0.72	0.82
CAIA	HBOS	0.45	0.27	0.27	0.02	0.47	0.45
	LOF	0.21	-0.05	0.18	-0.10	0.41	0.47
	kNN	0.18	-0.10	0.06	-0.26	0.41	0.47
	iForest	0.31	0.08	0.21	-0.06	0.47	0.56
	SDO	0.32	0.09	0.45	0.26	0.52	0.60
AGM	HBOS	0.03	0.03	0.04	0.03	0.10	0.92
	LOF	0.01	0.01	0.03	0.02	0.02	0.63
	kNN	0.13	0.13	0.20	0.20	0.13	0.81
	iForest	0.04	0.04	0.05	0.04	0.09	0.91
	SDO	0.00	-0.00	0.00	-0.00	0.09	0.95
TA	HBOS	0.03	0.03	0.04	0.04	0.03	0.53
	LOF	0.00	0.00	0.00	-0.00	0.01	0.53
	kNN	0.04	0.03	0.05	0.05	0.04	0.58
	iForest	0.03	0.03	0.03	0.02	0.04	0.54
	SDO	0.04	0.04	0.07	0.07	0.05	0.54
Cisco	HBOS	0.02	-0.20	0.01	-0.21	0.32	0.26
	LOF	0.09	-0.12	0.15	-0.04	0.32	0.28
	kNN	0.01	-0.21	0.01	-0.21	0.31	0.12
	iForest	0.02	-0.20	0.01	-0.21	0.33	0.27
	SDO	0.02	-0.20	0.01	-0.21	0.52	0.65

also for this vector and obtained the following values: kNN, $k = 15$; LOF, $MinPts = 50$; HBOS, $k = 22$; iForest, $t = 50, f = 4, \psi = 456$; SDO, $k = 241, x = 25$, $q_v = 0.35$. Some histograms are shown in Figure 3.12.

Obtained outlierness box plots are shown in Figure 3.9, histograms in Figure 3.12, and performance indices in Table 3.7. Results disclose that the OptOut vector considerably increases performances and, therefore, the capability of algorithms to discriminate attacks based on outlierness (particularly when using SDO). However, real-world detection demands high accuracy to minimize the proliferation of false positives. Attack detection based on unsupervised algorithms can hardly solve the problem alone, but its combination with supervised methods and techniques that leverage pre-knowledge is expected to build detection frameworks with highly effective performances.

Table 3.6: OptOut feature vector after forward selection (SDO nested).

Original vector	Feature	Description
AGM	pkts_mode_dstIP	Packets received by the most common IP destination.
AGM	#TTL	Number of different Time-to-Live (TTL) used by the IP source.
CAIA/Consensus	srcPkts	Packets sent by the IP source.
CAIA	stdev_dstPktLength	Standard deviation of the length of the packets sent by the IP destination.
CAIA/Consensus	srcBytes	Bytes sent by the IP source.

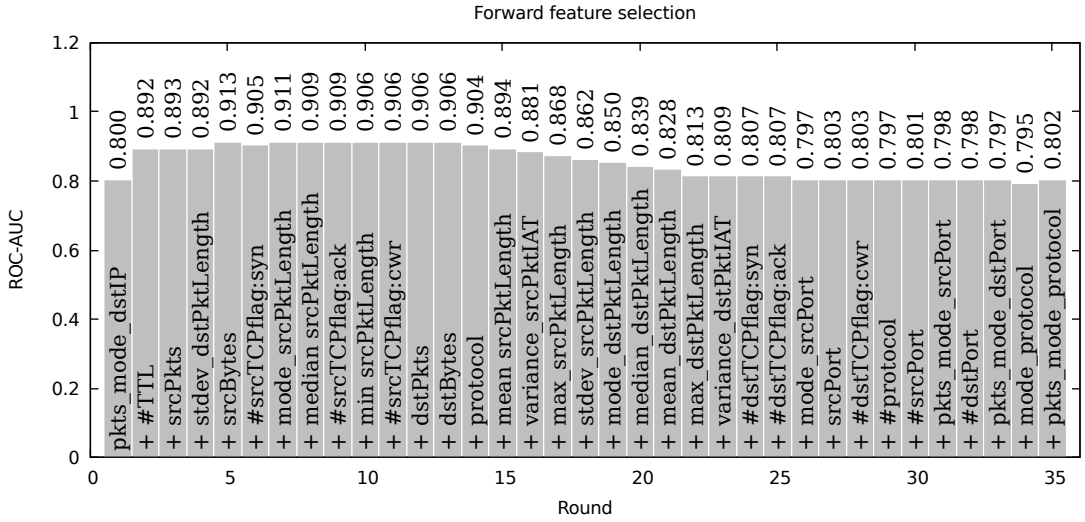


Figure 3.11: OptOut forward selection process.

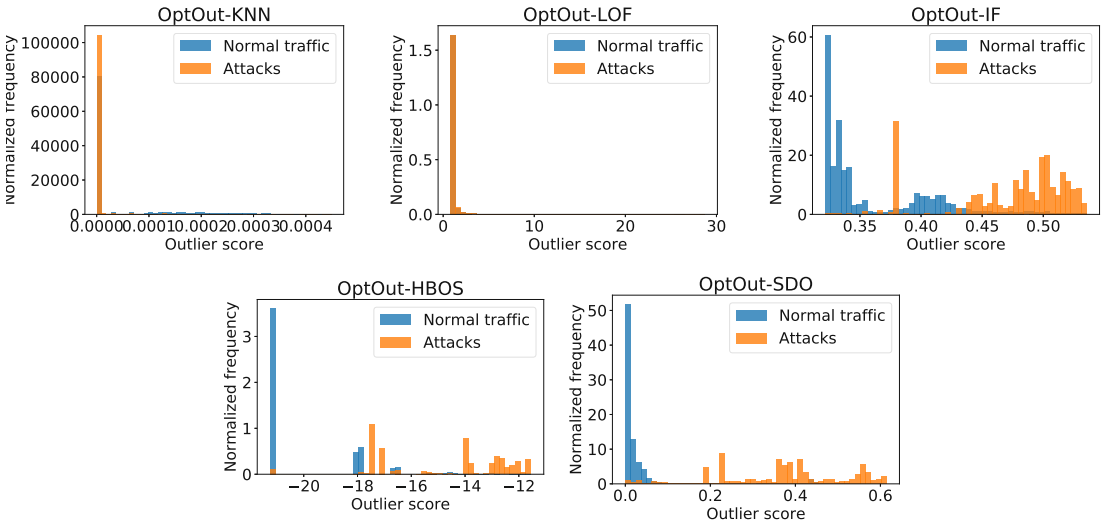


Figure 3.12: OptOut vector. Normalized histograms (top 5% outliers removed for a better visualization).

Table 3.7: Algorithm performances for the OptOut feature vector.

	P@n	AP@n	AP	AAP	Max. F1	ROC-AUC
HBOS	0.74	0.65	0.93	0.90	0.87	0.96
LOF	0.20	-0.07	0.20	-0.07	0.41	0.46
kNN	0.09	-0.22	0.17	-0.11	0.40	0.43
iForest	0.78	0.70	0.72	0.63	0.79	0.92
SDO	0.90	0.87	0.97	0.96	0.92	0.98

3.3.5 Discussion

In this section, we have faced three relevant aspects of network attacks, namely: (a) if they are actually outliers, (b) what are the most suitable algorithms and feature vectors for implementing outlierness-based detectors, and (c) if the attack outlierness is enough for implementing real-world detection. We have studied these questions from analytical perspectives by evaluating five different feature vectors used in the literature with five different outlier-ranking algorithms. For our experiments, we have used a dataset for NID evaluation that reflects modern attacks as well as legitimate behavior profiles.

The conducted experiments reveal that, as a general rule, network attacks have higher global distance-based outlierness averages than normal traffic. Given the characteristics of network feature spaces – noisy, highly varied, with normal instances covering a broad spectrum and drawing subspaces with many density differences – local algorithms show low performances for attack detection. Algorithms with a more global space interpretation – like SDO or HBOS – tend to perform better, particularly when representation spaces capture the behavior of network devices and hosts (e.g., the AGM format). We have proposed a feature vector that maximizes the separation of attacks and non-attacks in terms of outlierness. However, the risk of high false positive rates still prevails due to the base-rate fallacy problem inherent to network security spaces. Outlier detection algorithms can be a powerful tool for detecting known and novel attacks, but leveraging pre-knowledge with supervised methods should not be omitted, since supervised and unsupervised methods are complementary and, together, can build highly refined solutions.

CHAPTER 4

Novel Analysis Methods

Despite a variety of methods that have been proposed in the field of streaming anomaly detection, the previous chapter has shown that anomaly detection methods that qualify for being used in network attack detection are hardly available. Explainability is a major concern for such applications, which narrows the range of deployable methods to distance-based algorithms. Distance-based algorithms allow a straightforward interpretation of outlierness based on distances to neighboring samples. On the other hand, due to the large amount of samples that have to be processed, computational speed must be considered and computational complexity that is higher than linear is likely to be prohibitive.

Furthermore, a peculiarity of benign network traffic is that it in many cases shows strong temporal patterns due to the diurnal and weekly rhythm of Internet usage. For successful attack detection, it would therefore be useful to leverage this information by detecting anomalous traffic in a spatiotemporal sense that captures periodicities, instead of merely spotting traffic that is different from any previously seen data. In this chapter, we therefore devise techniques to cope with the challenges described above.

Finally, however, modern encryption techniques might complicate the task of analyzing network traffic. If several flows are jointly transmitted over a single encrypted link, information of individual flows cannot be separately fed into an ML classifier. At the same time, equipping an IDS with the ability to decrypt is likely to constitute a security risk by itself. It is therefore a relevant question whether individual flows can be analyzed in encrypted data purely based on known patterns in traffic data. At the end of this chapter, we also investigate this question using a deep learning-based approach.

We thus cover important aspects of our postulated research questions in this chapter. Based on our results concerning RQ1 in the previous Chapter 3, a main goal is the development of an algorithm that is able to capture the characteristics of NTA. We therefore address RQ3, since explainability is a main goal of an appropriate algorithm. Furthermore, however, also the analysis of encrypted traffic prompts for development of

new methods for dealing with encrypted data. We therefore also address RQ4 in this chapter to approach the question whether even encrypted VPN traffic can be analyzed using ML methods.

4.1 Related Work and State of the Art

We refer to the previous Chapter 3 for a detailed overview of methods for detecting anomalous points in data streams. Of particular interest are methods that provide interpretable results, which applies mainly to distance-based methods. In this section, we will therefore provide additional related work related to the construction of algorithms we introduce in this chapter. Several fields of research touch the problem settings that we explore in this chapter, but differ in essential aspects.

4.1.1 SDO

In this thesis, we extend the SDO outlier detection algorithm [127] for outlier detection on static datasets that is inspired by the kNN method. We have already provided a brief overview of SDO in Section 3.3.1 and now provide a more detailed description. SDO builds a low-density model for a dataset by randomly sampling $k \in \mathbb{N}$ points from it, which are termed *observers*. To avoid using outliers as observers, the set of observers is cleaned based on a quality metric. An outlier score is obtained as the median distance to the $x \in \mathbb{N}$ closest observers.

Hence, in a nutshell the SDO algorithm can be summarized as follows:

1. Randomly sample observers from the dataset.
2. For each observer, compute the observer's *observations* as the number of points lying in the observer's neighborhood.
3. Partition the observers set into *idle* and *active* observers based on an observations threshold.
4. Compute an outlier score as median distance to closest active observers.

The data are thus down-sampled to the observers set, containing only representative space locations and idle observers are avoided for outlier scoring to clean the model from points that might be outliers themselves.

In this thesis, we devise SDOstream, an algorithm for performing outlier detection on data streams that is based on SDO. The intuition of the notion of observers also holds for SDOstream. However, we will adapt the algorithm for a streaming setting by continuously sampling new points as observers and using an EWMA operation for computing observations.

4.1.2 Time Series Analysis

A time series is a temporal sequence of observations of specific measurement variables. Time series have been used and analyzed in various fields of research, e.g., finance and econometrics [147, 111, 160, 197], weather forecasting [192, 160, 197], electric load forecasting [197] or signal processing [159]. Traditionally, time series have been processed and analyzed using strict mathematical tools. For an introduction, the reader is referred to available text books on the topic [111, 67]. However, also data mining and ML is applied to time series, for example, in forecasting [160, 197], clustering [153], anomaly detection [151] or change point detection [29].

When processing time series, temporal patterns and dependencies are assumed to exist in the observed data. In contrast to this assumption, in this thesis we understand data as being generated by a number of sources that exhibit diverse temporal patterns. Therefore, data points arrive to the observation location in a mixed up fashion, destroying or hiding any pattern of individual clusters when analyzing the data as a whole. Hence, instead of the observed data as a unit, it is the points arriving at a specific location in the feature space, which we assume to independently or in groups draw time series that can be analyzed for temporal dependencies.

4.1.3 Coreset Algorithms

The sampling-based approach of our method is related to the concept of *coresets*, introduced by Agarwal et al. in [17]. Coresets aim to solve a problem on a small set of representative points instead of solving it on the complete dataset. While authors give different definitions of what exactly a coreset is [17, 87, 40, 125], a property usually emphasized is that the solution on the coreset approximates the solution evaluated on the full dataset up to a defined error.

Coresets have been proposed for approximating geometric extent measures [17], nearest-neighbor classification [87], k-means clustering [40] and Bayesian inference [125].

4.1.4 Periodic Pattern Mining

The problem of detecting periodicities and temporal patterns in sequences of data has also been investigated in the context of periodic pattern mining [90, 89]. Here, sequences with discrete time steps are analyzed to identify recurring patterns in a sequence of events from a finite set.

Periodic pattern mining can be applied to spatiotemporal data [240] to detect periodicities in the movement of objects. In contrast to this approach, in this section we focus on detecting periodicities of arbitrary clusters even if the corresponding data points are mixed up with data points from other clusters that exhibit different temporal patterns or no patterns at all. To the best of our knowledge, this problem setting has not been explored before.

4.1.5 Analysis of Encrypted Traffic

Finally, we also investigate the analysis of encrypted traffic in this chapter. Analysis and classification of network traffic is a well-known research area. A substantial body of research exists on the analysis of encrypted traffic (e.g., [187, 27, 22, 24, 23, 25, 26, 214, 175, 37, 148]). We refer to survey papers [221, 64] on this subject for a comprehensive overview. While these papers demonstrate that classification of encrypted flows is possible even when only meta information like packet sizes and IATs are known, it is common to the majority of existing research to assume that the separation into flows has been done in advance. This is a reasonable assumption when encryption is done only on the transport layer, but is unable to handle more comprehensive encryption techniques we target in this thesis.

Some research has been conducted on deanonymization of VPN traffic. Appelbaum et al. [35] outline several strategies an attacker might use to deanonymize a victim's VPN traffic, distinguishing attackers based on their position and on their capabilities. Unlike targeting the patterns of the encrypted traffic itself, the paper reveals several scenarios for how an attacker can make use of leaking information. Similarly, Bui et al. [60] highlight shortcomings in the client configuration of commercial VPN providers that might allow attacking the VPN's encryption.

Of particular interest to the research community are deanonymization attacks on the Tor network [195, 21, 139, 43], since Tor aims to achieve the very purpose of providing strong anonymity. The papers [139, 43] provide a recent overview of approaches to deanonymize Tor traffic. Besides attacks that are based on leaking information through side channels and active attacks and attacks that are based on actively exploiting shortcomings of client applications, also several passive attacks on Tor have been proposed. However, these passive attacks are usually based on associating the exit nodes' connections with the users' connections by leveraging attacker-controlled Tor nodes. This scenario is very different from the setting we consider here.

Unlike previous research, we base our analysis on observed encrypted traffic directly, even if it consists of packets from multiple interleaved flows. A related research effort has been conducted by Meghdouri et al. [167], who showed that by using a deep NN, it is possible to accurately identify the number of flows contained in an encrypted VPN tunnel, hence disclosing information about the encrypted traffic. In this thesis, we go one step further and focus on the problem of separating packets in encrypted tunnel traffic into their respective flows. To the best of our knowledge, this problem has not been investigated before.

4.2 SDOstream

Notice of adoption from previous publications (Section 4.2)
Parts of the contents of this section have been published in the following papers:

[117] Alexander Hartl, Félix Iglesias, and Tanja Zseby. SDOstream: Low-density models for streaming outlier detection. In ESANN 2020 proceedings*, pages 661–666, 2020*

Academic work has been predominantly carried out by myself. All authors contributed in discussion of experiments and in proofreading of the manuscript.

We first turn to the problem of designing an outlier detector for streaming data that is well suited for the characteristics of network data, thus addressing RQ3 and building upon our findings from the previous chapter answering RQ1. In stream data processing, data points $\boldsymbol{v}_j \in \mathbb{R}^D$ arrive continuously at monotonically increasing, but otherwise arbitrary times $t_j \in \mathbb{R}$ for $j = 1, 2, \ldots$. Due to the steady arrival of new data, knowledge discovery and data mining tasks pose various challenges in this setting, as in many fields of research and engineering data accumulates to a substantial volume within short time. In NTA, but also frequently in other applications, algorithms for anomaly detection on data streams are required to operate in an online fashion, allowing to perform classifications whenever new points arrive, which makes expensive retraining procedures impossible. At the same time, however, *concept drift* demands to continuously adapt the model to newly seen patterns and forget old, outdated ones.

Some alternatives to implement adaptiveness are re-fitting or updating models. However, anomaly (or outlier) detection is traditionally faced using instance-based methods like kNN or LOF [58]. As general purpose options, such methods have not been overcome yet in terms of accuracy [63]. On the other hand, their main drawback is the need to consider all previous data for every new data point, a fact that significantly slows down the analysis. For streaming data, this problem is partially alleviated by using a SW, which can be considered a memory length. Moreover, *outlierness* is strictly defined based on non-intuitive parameters, many times being arbitrarily adjusted.

We introduce SDOstream, a simple and elegant outlier detection algorithm for evolving data streams, which operates in $O(n)$ in the number of processed data points. SDOstream builds a model for the data and, hence, avoids the need to store all points in a SW. SWs define a cut-off time, before which data points are not remembered. In SDOstream, impact of past data points is of exponential shape. Hence, it has the potential to retain shapes for a much longer time period. Therefore, whereas in SWs memory length frequently has to be decided based on algorithm limitations, in SDOstream it is defined only based on the application.

Furthermore, in SDOstream the final definition of outlierness is established based on space distances and representative point locations, providing an intuitive and interpretable decision process.

Compared to the above methods for streaming data, SDOstream benefits from operating with a model, but avoid drawbacks emerging from expensive model refitting procedures. It is not limited by a SW, so it avoids suffering from *short-sightedness* (Figure 4.1). It

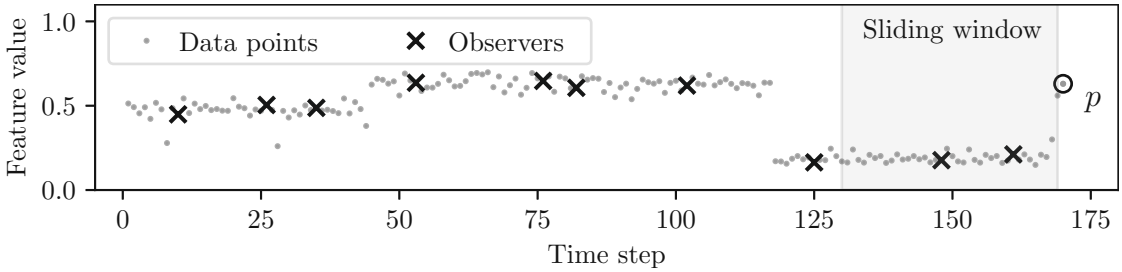


Figure 4.1: Short-sightedness of SWs: p is an outlier for the SW, but an inlier for SDOstream.

operates in linear time like RFs, but by keeping space locations in the model, its definition of outlierness preserves data shapes instead of solely the trained model.

4.2.1 Notation

In this chapter, we denote by $[N_{\text{bins}}]$ with $N_{\text{bins}} \in \mathbb{N}$ the set $\{0, \ldots, N_{\text{bins}} - 1\}$ and by $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}_0^+$ a distance function like, e.g., the Euclidean distance. To ease comprehension, we present an overview of most important symbols and notation used throughout Sections 4.2 and 4.3 in Table 4.1. We refer to later sections for explanation of terms used in Table 4.1.

4.2.2 Algorithm Design

The SDO algorithm can be adapted for streaming operation in a very natural way. We can adopt the usage of a fixed-size set of data points as observers, representing a model, consider a fixed-size fraction of observers as idle and compute an outlierness score as median of distances to the x closest active observers. Streaming operation requires permanent adjustment of the model to new data. Hence, we introduce additional techniques to update observer quality metrics and observers themselves as new data arrives.

Parameters

The most important parameter of SDOstream is $f \in (0, 1)$, the fading parameter. f controls how quickly the model is able to adjust to newly observed data clusters and, on the other hand, how stable the model is with respect to noise in the processed data. It is beneficial to write f as $f = \exp(-T^{-1})$ with a time parameter $T \in \mathbb{R}^+$, which in its function can be best understood as similar to the window size of SW approaches. Furthermore, $k \in \mathbb{N}$ denotes the model size, i.e. the number of observers to use in stationary operation. Intuitively, k should be increased for highly diverse data. Finally, $q_{\text{id}} \in (0, 1)$ denotes the fraction of observers to ignore for outlier scoring (called *idle* observers) and $x \in \mathbb{N}$ is the number of nearest observers to consider. The reader is referred to [127] for a discussion of how to choose these parameters.

Table 4.1: Symbols and notation.

Algorithm Parameters	$k \in \mathbb{N}$	Number of observers.
	$x \in \mathbb{N}$	Number of nearest observers.
	$f \in (0, 1)$	Fading parameter.
	$T = -1/\ln f$	EWMA time constant.
	$T_0 \in \mathbb{R}^+$	Fourier Transform (FT) base period.‡
	$N_{\text{bins}} \in \mathbb{N}$	Number of frequency bins.‡
	$q_{\text{id}} \in [0, 1]$	Observer idle-active fraction.
Algorithm State	Ω	Observers set.
	$P_\omega \in \mathbb{R}^+$	Observations by ω.†
	$P_{\boldsymbol{\omega},n} \in \mathbb{C}$	Fourier coefficients for $\boldsymbol{\omega}$'s observations.‡
	$H_{\boldsymbol{\omega}} \in \mathbb{R}^+$	Reference for age-normalization of $P_{\boldsymbol{\omega},0}$.
	$\overline{P}_\omega = P_\omega/(1 - f^{H_\omega})$	Average observations by ω.
	$i_{\text{LAO}} \in \mathbb{N}$	Index of last added observer.
Further Notation	$\boldsymbol{\omega} \in \Omega$	An observer.
	$d(\cdot, \cdot)$	A distance function.
	$\mathcal{N} \subset \Omega$	Set of nearest observers.
	$\mathcal{N}_a \subset \Omega$	Set of nearest active observers.
	$n \in [N_{\text{bins}}]$	The frequency index.‡

† Only applies to the plain SDOstream algorithm outlined in Section 4.2.
‡ Only applies to SDOstream's extension for periodic patterns outlined in Section 4.3.

Observer Idle-Active Split

In SDO, observations $P_\omega \in \mathbb{N}$ for an observer ω, i.e. the number of data points for which ω is contained in the nearest-observers set, constitutes a quality metric, which is used as basis for determining the set of active observers. To adjust P_ω to newly seen data, while being able to scale to arbitrary time scales, we deploy an exponential moving averaging approach. In particular, we keep track of an exponential moving average $P_\omega \in \mathbb{R}^+$ of observations by ω. Hence, for each processed data point, we set $P_\omega \leftarrow fP_\omega + 1$ if it belongs to the nearest-observers set of the current data point, and $P_\omega \leftarrow fP_\omega$ if it does not. We use P_ω to distinguish between active and idle observers, i.e., the q_{id}-fraction of observers with the lowest P_ω values are declared idle.

Replacing Observers

To retain an up-to-date model, it is necessary to regularly replace outdated observers by new data points. It appears reasonable to select observers for removal based on the minimization of a quality metric like P_ω. However, new observers have a lower P_ω than

established observers. While this is intended for the idle-active split, selecting observers for removal based on P_ω is ill-fated, as it would lead to new observers being replaced constantly. Instead, an average observer quality $\overline{P}_\omega \in \mathbb{R}^+$ is required, which sets P_ω in relation to the maximum value it can assume. Assume that data points arrive with a constant IAT of $\delta \in \mathbb{R}^+$. Then, after a time $H \in \mathbb{R}^+$, i.e. after $\lfloor \frac{H}{\delta} \rfloor$ data points, the maximum P_ω can have assumed is $1 + f^\delta + \ldots + f^{\lfloor \frac{H}{\delta} \rfloor \delta} = \frac{1-f^{(\lfloor \frac{H}{\delta} \rfloor + 1)\delta}}{1-f^\delta} \approx \frac{1-f^H}{1-f^\delta}$. Hence, $P_\omega / \frac{1-f^H}{1-f^\delta}$ can be used as a metric for selecting the observer to replace. Since we are only interested in comparing observers, the constant factor $1-\delta$ can be omitted, obtaining $\overline{P}_\omega = P_\omega/(1-f^H)$. Clearly, real-world IATs are not constant. However, it can be shown that the provided $\overline{P}_\omega$ remains valid also for non-constant IATs, given that the average IAT does not change at a time scale of model adjustment time.

Putting the Blocks Together

Algorithm 4.1 depicts SDOstream. Inputs consist of a data point $v_i \in \mathbb{R}^m$ and the corresponding time stamp $t_i \in \mathbb{R}$. In **line 1**, the x closest observers are determined from both sets of active and idle observers. Hence, the set of observers Ω is queried together with P_ω to determine the current idle-active split and find the sets $\mathcal{N}$ and $\mathcal{N}_a$, containing the x closest observers and x closest active observers, respectively.

In **lines 2-3** observer quality P_ω and age H_ω of all observers are adjusted and, subsequently, the model itself. For this purpose, in **line 4**, the probability of sampling a data point to be used as observer, is proportional to $\frac{\sum_{\omega\in\mathcal{N}} P_\omega}{\sum_{\omega\in\Omega} P_\omega}$ to support an even P_ω and, hence, a representative model. Furthermore, note that during initial start-up using all available data points as observers might lead to a very poor model due to temporal dependence in the data stream, e.g. if the k first seen data points all belong to the same cluster. Hence, also during start-up, observers are added with a finite rate, which is gradually decreased to quickly approach the desired model size on the one hand, and sample over a large

Algorithm 4.1: SDOstream: Processing a data point (v_i, t_i).

1: Find x closest observer sets $\mathcal{N}$ and $\mathcal{N}_a$
2: Set $H_\omega \leftarrow H_\omega + (t_i - t_{i-1})$ and $P_\omega \leftarrow f^{t_i - t_{i-1}} P_\omega \quad \forall\, \omega \in \Omega$
3: Set $P_\omega \leftarrow P_\omega + 1 \quad \forall\, \omega \in \mathcal{N}$
4: **if** Ω empty **or** $r \leq -\ln(f)\frac{k^2}{x}\frac{\sum_{\omega\in\mathcal{N}} P_\omega}{\sum_{\omega\in\Omega} P_\omega}\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}$ with $r \in_R [0,1]$ **then**
5: **if** $|\Omega| = k$ **then**
6: Remove $\arg\min_{\omega\in\Omega} \overline{P}_\omega$ from Ω
7: **end if**
8: Add v_i to Ω
9: Set $i_{\text{LAO}} \leftarrow i$, $P_{v_i} \leftarrow 1$ and $H_{v_i} \leftarrow 0$
10: **end if**
11: **return** $\text{median}_{\omega\in\mathcal{N}_a}(\|\omega - v_i\|)$ as outlier score

enough period on the other hand. In stationary operation data points are then sampled with a rate of $k/T = -k \ln f$, i.e. within a time $T = -1/\ln f$ all observers are replaced one time on average.

Finally, in **line 11**, outlier scoring takes place and, similar to SDO, is performed by computing the median distance to the x closest active observers.

4.2.3 Time and Space Complexity

If n denotes the number of processed samples, we show that SDOstream has space and time complexity $O(k)$ and $O(n \log k)$, respectively. Indeed, considering solely dependence on n, space and time complexity are $O(1)$ and $O(n)$, as SDOstream operates with a fixed-size model. This benefit allows scaling to data streams of arbitrary size.

Furthermore, Algorithm 4.1 suggests $O(nk)$ run time when considering also model size k. However, Algorithm 4.1 mainly aims at simplicity of presentation and for highly dynamic, diverse streams, sub-linear run time with respect to k might be desired.

To obtain run time in $O(n \log k)$, observers can be stored in a data structure for efficient nearest neighbor search like an M-Tree [70], for which the authors observed logarithmic dependence of tree size. Tree-based structures can be used for keeping observers ordered with respect to P_ω. Hence, both nearest observer search and idle-active split are possible in $O(\log k)$. Discovery of the observer with lowest $\overline{P}_\omega$ takes $O(k)$ and is required every $\frac{T}{k\delta}$th sample on average. Hence, SDOstream takes $O(n \log k + \frac{k\delta}{T} nk)$ amortized time. In the primary case of high-rate settings, i.e., $\delta/T \to 0$, total space and time complexity thus are $O(k)$ and $O(n \log k)$, respectively.

4.2.4 Performance Evaluation

We used real and artificial data to study SDOstream, using L_2 distance and deploying genetic algorithms for tuning hyperparameters for all studied algorithms. To investigate the temporal behavior for transient events like suddenly emerging clusters, we crafted a synthetic dataset with 100,000 data points, 6 clusters and 2% outliers using MDCGen [128]. We removed data points for two clusters, so that the clusters appear after the 50,000$^{\text{th}}$

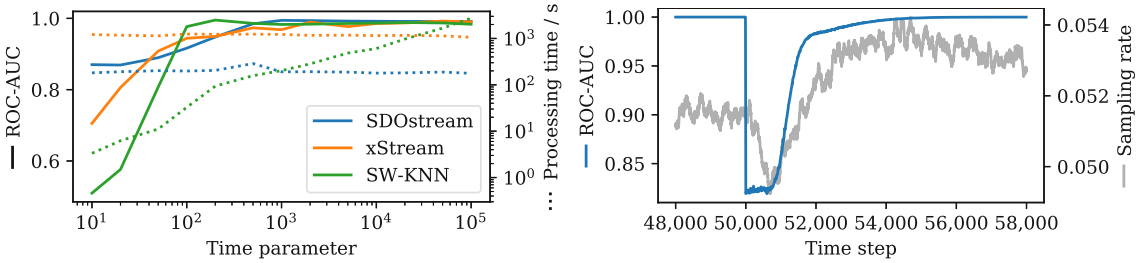


Figure 4.2: Performances for the KDDCup'99 dataset (left) and behavior for emerging new clusters at $t = 50,000$ with synthetic data (right).

Table 4.2: ROC-AUC scores for static datasets.

	SDOstream	SW-KNN	xStream	SDO	kNN	LOF	iForest
Annthyroid	0.627	0.594	0.522	0.610	0.644	0.674	0.807
Cardiotocogr.	0.815	0.808	0.810	0.818	0.784	0.810	0.804
PageBlocks	0.904	0.903	0.923	0.910	0.904	0.943	0.921

data point. On the right side, Figure 4.2 shows the ensemble ROC-AUC and the average sampling rate computed over 10,000 runs of SDOstream with $T = 10,000$ and appropriately randomly permuted datasets.

As shown, the clusters are first classified as outliers, but after 1,000 ($= \frac{T}{10}$) to 2,000 ($= \frac{2T}{10}$) data points, performance steeply increases, returning to stationary performance after about 4,000 ($\frac{4T}{10}$) data points. As a side effect of the factor $\frac{\sum_{\omega \in \mathcal{N}} P_\omega}{\sum_{\omega \in \Omega} P_\omega}$, the sampling rate is affected by the event to a very small extent.

On the left side of Figure 4.2 we provide a comparison of performance results for the KDDCup'99 dataset, which contains 4.9 million data samples of streaming data. The window size is used as time parameter for xStream and SW-KNN, and T is used for SDOstream. To evade influence of the convergence phase, we only used the second half of returned outlierness scores for computing the ROC-AUC for both hyperparameter search and performance evaluation. As depicted, SDOstream exhibits very good performance in streaming scenarios, retains low processing time and shows to be stable over a wide range of different T.

Since only few datasets are established containing streaming data, and to provide a better comparison with popular algorithms, Table 4.2 compares SDOstream also against non-streaming algorithms using static datasets from [63], again evaluating stationary performance by considering the second half of returned scores. Here, SDOstream shows performances comparable to established outlier detection methods.

Concluding, SDOstream proved to be a fast, versatile outlier detection algorithm, demonstrating its strengths particularly when facing substantial data volumes by creating a representative model of the data. While providing an intuitive outlierness definition, it is flexible and allows memory length to be set solely based on the application's requirements.

4.3 Mining Periodic Patterns

To date, research has yielded diverse approaches for anomaly detection on data streams, showing different qualities in coping with these challenges. However, disregarded by existing approaches, also the dynamics of the arrival process of data points themselves yield information that can be crucial for the success of a data-mining task. For instance, the appearance of certain clusters might be caused by a diurnal pattern while others occur at all times. Leveraging these temporal patterns is relevant for anomaly detection,

as it allows to spot *out-of-phase outliers*, i.e. data points that do not fit established temporal patterns. In addition, capturing and analyzing such temporal patterns found is also interesting for descriptive and knowledge extraction purposes.

In this section, we present an extension of our algorithm presented in the previous Section 4.2 for building and dynamically maintaining a fixed-size model from a stream of arriving data points. The generated model incorporates information about the time when points have been observed and, hence, the algorithm is able to capture periodical patterns in the data stream, while retaining constant per-sample space and time complexity. To fully escape dependence of the data volume, we continue to use an EWMA to estimate model information from the arriving data mass. Based on the theory of FTs, however, we now hold temporal information as coefficients of the functions $\exp(jk2\pi t/T_0)$ with $k \in \mathbb{N}_0$ and a base period T_0, which yield an orthogonal basis on any interval of length T_0. A view of the model representative for a specific time of interest t is finally formed by filtering the set of observers to contain only the most relevant ones for time t, inherently ignoring outliers, which would otherwise distort the learned model.

Our algorithm is a natural extension of SDOstream, adding temporal information to the model by lifting the EWMA operation to the complex plane. Having a model of what is normal allows us to state what does not conform to this model, inherently linking the task of anomaly and outlier detection to our method, but additionally considering temporal periodicities in the model of normality. From a data mining perspective, we consider the possibility of analyzing the steadily arriving data stream for spatial and temporal patterns as an equally important contribution as providing outlier detection for stream data. We provide experimental results for both of these use cases.

In this chapter, we show relevant contributions for algorithmic outlier detection from several practical perspectives:

- *Scalability.* In a streaming setting, many applications process a vast amount of data points during a short period of time while, at the same time, requiring a memory length of weeks or even months to capture an accurate notion of normality. For many established outlier detection methods, space or time complexity increases substantially with memory length, which might become unfeasible or enforce the use of suboptimal parameters that downgrade the detection performance.

  With our work in this thesis, we target specifically data streams with high rates of arriving data points, e.g., several thousand per second. In particular, we assume the number of arriving data points within the configured memory length to exceed the configured model size. Since our method works with a fixed-size model, space and time complexity per data point do not depend on memory length. Furthermore, if a parameterization with a comprehensive model or a high temporal resolution is desired, our algorithm can use SIMD parallelism, which recently showed drastic improvements [215] due to the popularity of deep learning.

- *Interpretability.* Many modern outlier detection approaches use tree structures or random projections of feature spaces, which habitually improve detection at the cost of sacrificing model interpretability. This entails difficulties in understanding and explaining why the outlierness of a certain data point has been ranked as it has been. Interpretability of ML is key to extract knowledge, formulate hypothesis, and establish trust in many application fields, e.g., medicine [228, 150], judicial decisions [150] or NID [116].

  In our case, outlier scores are directly interpretable as deviations from closest observers. The analysis of the observers set is straightforward, as the number of observers is small, and is possible at any time, both online and in a forensic manner. Here, our proposed algorithm also provides temporal information of observers' recurrence. In many cases, features are abstract and structures in high-dimensional feature spaces are difficult to grasp. Hence, by adding temporal information to the model, our proposed method adds substantial value to interpretability.

- *Accuracy.* Capturing the temporal occurrence of clusters provides our method with accuracy benefits for outlier detection as it enables the detection of out-of-phase outliers, which would go undetected by traditional outlier detection methods. We show that our algorithm is able to keep up with or outperform state-of-the-art performance on publicly available datasets for outlier detection evaluation.

The remainder of this section is structured as follows. After summarizing the goals of our new approach, we introduce the concept of out-of-phase outliers. We will then describe our method and validate our proposal twofold: from a proof of concept perspective, by showing that out-of-phase outliers can be reliably detected, and with performance evaluations on real-world data and publicly available benchmarking datasets for outlier detection, in which our algorithm achieves or surpasses state-of-the-art results, and additionally provides a useful characterization of the streaming data.

4.3.1 Preliminaries

Observers-based Outlier Detection: Notation and Recap

Similar to the previous Section 4.2, we denote the observers set as Ω and, accepting a slight abuse of notation, we denote by $\boldsymbol{\omega} \in \Omega$ both an abstract observer and its feature vector. Furthermore, $P_{\boldsymbol{\omega}}$ denotes $\boldsymbol{\omega}$'s observations, where $P_{\boldsymbol{\omega}} \in \mathbb{N}_0$ for SDO and $P_{\boldsymbol{\omega}} \in \mathbb{R}_0^+$ for SDOstream. Hence, $P_{\boldsymbol{\omega}}$ counts the number of data points for which $\boldsymbol{\omega}$ belongs to the x nearest observers with an algorithm parameter $x \in \mathbb{N}$. Observers with insufficient $P_{\boldsymbol{\omega}}$ are thus disregarded for outlier scoring.

Core Novelty. Our extended algorithm replaces the scalar number of observations $P_{\boldsymbol{\omega}}$ with a temporal function, allowing active observers to become idle and reappear dynamically in accordance with the temporal pattern of the underlying clusters. Therefore, at any time it is possible to construct an active observers set representative for the data stream at the current time, opening up various use cases for further processing. By

basing outlier detection on the active observers set, it is possible to detect data points that do not meet the established temporal pattern, constituting outliers that might go undetected with traditional outlier detection approaches.

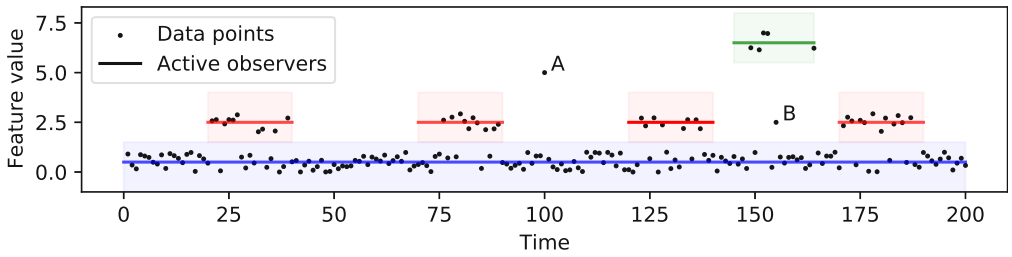


Figure 4.3: Illustration of out-of-phase outliers. While A is a spatial outlier, B is a out-of-phase outlier.

Out-of-Phase Outliers

Besides the usual spatial outlier, which indicates a data point that deviates from previously seen points in a purely spatial sense, traditional stream outlier detection algorithms are able to detect temporal outliers in terms of concept drift. Hence, after a pre-fixed time period, observed shapes are forgotten and points that would have been considered inliers are detected as temporal outliers.

However, control processes and human-related activities involve regularly repeating activities, habits and events, governed by, e.g., a diurnal or weekly rhythm. As a consequence, when analyzing clusters in data resulting from this behavior, we can identify times when appearances are likely and times when they are less likely. Hence, besides the usual spatial and temporal outliers, in this thesis, we acknowledge out-of-phase outliers and propose a method able to detect them. Figure 4.3 illustrates this distinction. While the spatial outlier A deviates from all previous points, the out-of-phase outlier B would be considered as inlier if it occurred within the cluster period.

Figure 4.3 also illustrates the functioning of our method in a nutshell. In the example, data points draw three clusters that occur with different dynamics. The algorithm captures the three clusters in the model with three observers that, based on previously seen data, are expected to be located in the neighborhood of newly arriving data. Observers also keep temporal information about cluster appearance by learning from when points are seen in an observer's neighborhood. They keep such information in frequency space, which yields the benefits of (1) being in line with the EWMA operation and (2) automatically performing a temporal interpolation, which would otherwise be necessary as data points can arrive at arbitrary real-valued times.

4.3.2 Our Method

We now describe the construction of our proposed method. To this end, we denote by $[N_{\text{bins}}]$ with $N_{\text{bins}} \in \mathbb{N}$ the set $\{0, \ldots, N_{\text{bins}} - 1\}$ and by $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^+$ a distance

function like, e.g., the Euclidean distance. To ease comprehension, we present an overview of most important symbols and notation used throughout this chapter in Table 4.1.

In contrast to previous work, our method enables the model to absorb information about temporal patterns in processed data streams. To describe this use case, we consider data streams satisfying the following definition.

Definition 1. *For a given data stream, let $\gamma(\boldsymbol{v}, t)$ denote the expected rate of arriving data points at location $\boldsymbol{v} \in \mathbb{R}^D$ and time $t \in \mathbb{R}$, i.e. $\gamma(\boldsymbol{v}, t)\Delta\boldsymbol{v}\Delta t$ denotes the expected number of data points seen in a volume $\Delta\boldsymbol{v}$ and duration Δt. We say that the stream exhibits T_0-periodic patterns with $T_0 \in \mathbb{R}^+$ if $\gamma(\boldsymbol{v}, t)$ is T_0-periodic, i.e., $\gamma(\boldsymbol{v}, t) = \gamma(\boldsymbol{v}, t + T_0)$ for all $\boldsymbol{v} \in \mathbb{R}^D, t \in \mathbb{R}$.*

It is important to note that we base Definition 1 on the *expected* rate of arriving data points. In many practical scenarios involving real-valued streams, data points that have been seen in the past never reoccur with the exact same feature vector, but only points in the points' vicinity can be reobserved. Hence, we model the stream as being generated by an underlying deterministic process to be able to reason about periodic behavior at a certain location.

Note, in particular, that a stationary stream exhibits T_0-periodic patterns for any T_0. Furthermore, while we describe the method based on Definition 1, the definition does not include concept drift, which is additionally tackled by the method in terms of EWMA.

To capture temporal patterns, we allow observers' observations to be T_0-periodic. We represent and store the associated temporal functions in terms of their FT coefficients $P_{\boldsymbol{\omega},n} \in \mathbb{C}$ with $\boldsymbol{\omega} \in \Omega, n \in [N_{\text{bins}}]$. To extract information relevant for the current point in time from the model, we define the q_{id}-percentile $P_{\text{thr}} \in \mathbb{R}^+$ of the observers' average observations,

$$P_{\text{thr}} = \max\left\{\rho \in \mathbb{R} \,\middle|\, \left|\{\boldsymbol{\omega} \in \Omega \,|\, P_{\boldsymbol{\omega},0} < \rho\}\right| \leq q_{\text{id}}|\Omega|\right\}, \tag{4.1}$$

and construct a view yielding the currently most relevant, i.e., *active* observers

$$\Omega_a = \left\{\boldsymbol{\omega} \in \Omega \,\middle|\, \Re\left\{\sum\nolimits_{n\in[N_{\text{bins}}]} P_{\boldsymbol{\omega},n}\right\} \geq P_{\text{thr}}\right\} \tag{4.2}$$

in terms of a lower-bound for the inverse FT of observers.

To narrow the scope to the most relevant information in a spatiotemporal sense, for a point $\boldsymbol{v} \in \mathbb{R}^D$ we then specify the set of nearest observers $\mathcal{N}(\boldsymbol{v}) \subset \Omega$ with $|\mathcal{N}| = \min(x, |\Omega|)$ and the set of nearest *active* observers $\mathcal{N}_a(\boldsymbol{v}) \subset \Omega_a$ with $|\mathcal{N}_a| = \min(x, |\Omega_a|)$, so that

$$d(\tilde{\boldsymbol{\omega}}, \boldsymbol{v}) \leq d(\boldsymbol{\omega}, \boldsymbol{v}) \,\forall\, \tilde{\boldsymbol{\omega}} \in \mathcal{N}(\boldsymbol{v}), \boldsymbol{\omega} \in \Omega \setminus \mathcal{N}(\boldsymbol{v}) \text{ and} \tag{4.3}$$

$$d(\tilde{\boldsymbol{\omega}}, \boldsymbol{v}) \leq d(\boldsymbol{\omega}, \boldsymbol{v}) \,\forall\, \tilde{\boldsymbol{\omega}} \in \mathcal{N}_a(\boldsymbol{v}), \boldsymbol{\omega} \in \Omega_a \setminus \mathcal{N}_a(\boldsymbol{v}). \tag{4.4}$$

Algorithm 4.2 depicts our core algorithm. We will discuss details on algorithm construction in the following sections.

Algorithm Construction

From a functional perspective, Algorithm 4.2 can be divided into the three parts, which are concerned with establishing the set of active observers Ω_a (**line 1**), using it for outlierness scoring (**line 2**), and updating the model (**lines 3-10**). The core concept, which allows capturing periodical patterns, is based on Lemma 1.

Lemma 1. *For an observer $\boldsymbol{\omega} \in \Omega$, let $g(t) \in \mathbb{R}^+$ denote the expected rate of arriving data points, for which $\boldsymbol{\omega}$ is contained in $\mathcal{N}$ at time t. If $g(t)$ is a T_0-periodic function and $T \gg T_0$, observations $P_{\boldsymbol{\omega},n}$ approximate a FT $E\{P_{\boldsymbol{\omega},n}\} \approx \int_{-T_0}^{0} g(\tau - t) \exp(-j2\pi n\tau/T_0) d\tau$ up to a constant factor.*

Proof. Let $i_o \in \mathbb{N}$ denote the index of a data point, for which $\boldsymbol{\omega}$ is contained in $\mathcal{N}$ and i_c the index of the currently processed data point, i.e. $i_o < i_c$. Then, the contribution of i_o to $P_{\boldsymbol{\omega},n}$ according to **line 4** of Algorithm 4.2 has been multiplied by $\Pi_{i=i_o+1}^{i_c} \big(\exp(-T^{-1} + jn2\pi/T_0)\big)^{t_i - t_{i-1}} = \big(\exp(-T^{-1} + jn2\pi/T_0)\big)^{t_{i_c} - t_{i_o}}$. Summing over all points that have arrived in $\boldsymbol{\omega}$'s neighborhood, we can write

$$E\{P_{\boldsymbol{\omega},n}\} = \int_{-\infty}^{t} g(\tau) \Big(\exp(-T^{-1} + jn2\pi/T_0)\Big)^{t-\tau} d\tau.$$

Splitting the integral into intervals of length T_0, we obtain

$$\begin{aligned} E\{P_{\boldsymbol{\omega},n}\} = & \sum_{l=0}^{\infty} \int_{t-T_0}^{t} g(\tau - lT_0) \big(\exp(-T^{-1} + jn2\pi/T_0)\big)^{t-\tau-lT_0} d\tau = \\ & \Big(\sum_{l=0}^{\infty} \exp(-T^{-1} lT_0)\Big) \int_{t-T_0}^{t} g(\tau) \big(\exp(-T^{-1} + jn2\pi/T_0)\big)^{t-\tau} d\tau \end{aligned}$$

Algorithm 4.2: Processing a data point $(\boldsymbol{v}_i, t_i)$ for outlier detection and temporal pattern discovery.

1: Find x closest observer sets $\mathcal{N}(\boldsymbol{v}_i)$ and $\mathcal{N}_a(\boldsymbol{v}_i)$
2: **report** $\text{median}_{\boldsymbol{\omega} \in \mathcal{N}_a} d(\boldsymbol{\omega}, \boldsymbol{v}_i)$ as outlier score
3: Set $H_{\boldsymbol{\omega}} \leftarrow H_{\boldsymbol{\omega}} \exp(-\frac{t_i - t_{i-1}}{T}) + 1 \quad \forall \boldsymbol{\omega} \in \Omega$
4: Set $P_{\boldsymbol{\omega},n} \leftarrow P_{\boldsymbol{\omega},n} \left[\exp(-\frac{1}{T} + \frac{jn2\pi}{T_0})\right]^{t_i - t_{i-1}} \quad \forall \boldsymbol{\omega} \in \Omega, n \in [N_{\text{bins}}]$
5: Set $P_{\boldsymbol{\omega},n} \leftarrow P_{\boldsymbol{\omega},n} + 1 \forall \boldsymbol{\omega} \in \mathcal{N}, n \in [N_{\text{bins}}]$
6: **if** $|\Omega| = 0$ **or** $r \leq \frac{k^2}{Tx} \frac{\sum_{\boldsymbol{\omega} \in \mathcal{N}} P_{\boldsymbol{\omega},0}}{\sum_{\boldsymbol{\omega} \in \Omega} P_{\boldsymbol{\omega},0}} \frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}$ with $r \in_R [0,1]$ **then**
7: Remove $\arg\min_{\boldsymbol{\omega} \in \Omega} \frac{P_{\boldsymbol{\omega},0}}{H_{\boldsymbol{\omega}}}$ from Ω **if** $|\Omega| = k$
8: Add $\boldsymbol{v}_i$ to Ω
9: Set $i_{\text{LAO}} \leftarrow i$, $H_{\boldsymbol{v}_i} \leftarrow 1$ and $P_{\boldsymbol{v}_i,n} \leftarrow 1 \forall n \in [N_{\text{bins}}]$
10: **end if**

---

due to T_0-periodicity of $g(t)$ and $\exp(jn2\pi) = 1$. Abbreviating the constant factor and substituting $\tau' = \tau - t$, we obtain

$$E\{P_{\boldsymbol{\omega},n}\} = c \int_{-T_0}^{0} g(\tau' - t)(\exp(-T^{-1} + jn2\pi/T_0))^{-\tau'} d\tau' \overset{T \gg T_0}{\approx} c \int_{-T_0}^{0} g(\tau' - t) \exp(-jn2\pi\tau'/T_0)) d\tau'. \qquad \square$$

Lemma 1 shows that temporal information about how frequently observers are used can be extracted from $P_{\boldsymbol{\omega},n}$ in terms of an inverse FT. To obtain a set of active observers Ω_a representative for the current point in time, it is natural to select observers that in the past have been used most often at corresponding points in time. Here, due to algorithm functioning observer usage is mainly considered at time $t - T_0$, which is reasonable due to T_0-periodicity. However, due to inherent interpolation also the very recent observer usage is considered, which is interesting particularly for new observers. In Theorem 1, we show that our method applies this approach for constructing Ω_a.

Theorem 1. *At time t, for data streams exhibiting T_0-periodic patterns, the active observers set Ω_a, as used by Algorithm 4.2, contains observers with highest $g(t)$.*

Proof. Equation 4.2 constructs the set Ω_a by selecting observers from Ω, for which $\Re\{\sum_{n\in[N_{\text{bins}}]} P_{\boldsymbol{\omega},n}\}$ is highest. If $P_{\boldsymbol{\omega},n}$ yields the FT of $g(t)$ according to Lemma 1, the theorem follows immediately, since $\Re\{\sum_{n\in[N_{\text{bins}}]} P_{\boldsymbol{\omega},n}\}$ performs the inverse FT at time $t = 0$ relative to the current time. $\square$

Theorem 1 allows us to make use of Ω_a for assessing spatiotemporal outlierness of arriving data points. Our method follows a nearest-observer approach for computing outlier scores. Hence, in **line 1**, $\mathcal{N}$ and $\mathcal{N}_a$ are constructed. We compute an outlier score based on the median of distances to the x closest observers.

Instead of using the median, the mean of distances to the closest observers might alternatively be used as reduction function. For our work, we base the decision to choose the median on the experiments performed in [128]. Superior results when using the median can be explained by increased robustness.

The final part of Algorithm 4.2 is concerned with updating the model to newly observed data, which involves replacing the least useful observer with a new data point. Updating of $P_{\boldsymbol{\omega},n}$ in **line 4** follows an exponential shape parameterized by time T. We show in Theorem 2 that replacing of observers proceeds with the same pace, which is necessary as observers otherwise would not be able to build up meaningful $P_{\boldsymbol{\omega},n}$ values.

Theorem 2. *For data streams exhibiting T_0-periodic patterns, Algorithm 4.2 on average samples k data points during a time period T as new observers.*

Proof. Taking **line 6** in Algorithm 4.2 as starting point, we have $\min\left(1, \frac{k^2}{Tx}\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}}{\sum_{\omega\in\Omega} P_{\omega,0}}\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}\right)$ as probability for sampling a given newly seen point as observer. Since we target specifically data streams with high rates of arriving data points, we can safely assume the probability of sampling a point to be small. Hence, $\Pr\left\{1 < \frac{k^2}{Tx}\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}}{\sum_{\omega\in\Omega} P_{\omega,0}}\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}\right\}$ is negligible and we can write for the average probability of sampling a new point as observer $P_s \approx E\left\{\frac{k^2}{Tx}\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}}{\sum_{\omega\in\Omega} P_{\omega,0}}\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}\right\}$. Under the same assumption, we observe that the term $\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}$ depends on the current time, but, since points belonging to different neighborhoods arrive in an interleaved manner, does not depend on a point's neighborhood. Since $\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}}{\sum_{\omega\in\Omega} P_{\omega,0}}$ does not depend on time, we can split the term to $P_s \approx E\left\{\frac{k^2}{Tx}\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}}{\sum_{\omega\in\Omega} P_{\omega,0}}\right\} E\left\{\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}\right\}$ due to stochastic independence of both terms. $\sum_{\omega\in\mathcal{N}} P_{\omega,0}/x$ expresses the average observation count in the current neighborhood. The algorithm implements several mechanisms to make the observer density agree with the time-averaged point density, rendering the time-averaged local average observation count $E\{\sum_{\omega\in\mathcal{N}} P_{\omega,0}/x\}$ equal to the total average observation count of all observers $\sum_{\omega\in\Omega} P_{\omega,0}/k$, hence $P_s \approx \frac{k}{T} E\left\{\frac{t_i - t_{i_{\text{LAO}}}}{i - i_{\text{LAO}}}\right\} = \frac{k}{T}\overline{\text{IAT}}$ with the average IAT of two data points $\overline{\text{IAT}}$. During a time period of T, $T/\overline{\text{IAT}}$ data points arrive, yielding an average number of sampled points of $P_s T/\overline{\text{IAT}} = k$. □

Note that the factor $\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}/x}{\sum_{\omega\in\Omega} P_{\omega,0}/k}$ might be omitted without invalidating Theorem 2. However, we include it to promote representativity of the observers set. Hence, underrepresented observers in a neighborhood cause the observation count in this neighborhood to increase, leading to a higher sampling probability, while overrepresented observers lead to a lower sampling probability in the same neighborhood.

Furthermore, in the transient starting phase, $\frac{\sum_{\omega\in\mathcal{N}} P_{\omega,0}/x}{\sum_{\omega\in\Omega} P_{\omega,0}/k}$ is larger compared to during usual operation because the observers set has not reached its full size yet. Hence, by incorporating such factor we make sampling higher during this transient phase, allowing the model to quickly reach its full size. On the other hand, selecting all observers from a very short timespan would be unlikely to result in a model that is representative for longer periods. Hence, the reciprocal dependency of sampling probability of model size yields a good tradeoff between quickly assuming the full model size and building the full model from a too short time span.

During time T the model is replaced one time on average according to Theorem 2. Since we use a fixed-size model, an observer has to be removed when adding a new one. Picking the removal candidate only based on its $P_{\omega,n}$ would give rise to new observers being constantly replaced, since new observers naturally show lower $P_{\omega,n}$ than older ones. Hence, we use an age-normalized observation count $\frac{P_{\omega,0}}{H_\omega}$ for selecting the observer to remove in **line 7**. By updating H_ω as depicted in **line 3**, H_ω denotes the maximum

$P_{\boldsymbol{\omega},0}$ an observer $\boldsymbol{\omega}$ can have assumed since it was added to the observers set. Thus, $\frac{P_{\boldsymbol{\omega},0}}{H_{\boldsymbol{\omega}}} \in [0,1]$, where the maximum value of 1 is assumed only if $\boldsymbol{\omega}$ has always been in $\mathcal{N}$ since it was added to the model.

Analyzing the Learned Model

When analyzing a data stream, we are interested in finding out at which points in time data points are seen in the data stream. Since in many practical scenarios behavior is different at different times with a certain, e.g., diurnal rhythm, this analysis has to include data points from a large enough period to capture behavior at different times accurately. In a high-rate setting, manual analysis for such behavior is highly difficult, since it might become difficult to even store enough data for later analysis to arrive at correct conclusions. Additionally, manual analysis in a practical real-valued feature space raises the question how to analyze for temporal behavior, since a point with a given feature vector need not ever reoccur even if the stream's behavior is periodic. Instead, the point density at a certain location should be considered for extracting temporal behavior.

At any point in time, the observers set Ω allows immediate access to a sampled set of observed data points, including also temporal information. The temporal shape $g(t)$ of observed data points in a neighborhood of a given observer $\boldsymbol{\omega}$ can be efficiently recovered in terms of an inverse FT,

$$g(t) = \Re\left\{\sum\nolimits_{n\in[N_{\text{bins}}]} P_{\boldsymbol{\omega},n} \exp(jtn2\pi/T_0)\right\}. \tag{4.5}$$

We understand the method as a part in a data processing pipeline. While Ω in many cases is small enough to be inspected manually, it can be subjected to, e.g., visualization or clustering techniques to extract further knowledge. In this regard, it is equally important to be able to analyze the entirety of observers, but also to analyze stream characteristics relevant only for the current point in time, which the active observers set Ω_a provides. Ω_a is used for scoring outlierness of arriving data points. Hence, by analyzing points in Ω_a the data analyst is able to come up with an interpretable explanation why a given outlying point has been marked as such.

Main Concepts and Properties

Algorithm 4.2 shows strong similarities to its predecessor SDOstream. In fact, our proposed method can be considered an extension of SDOstream. In contrast, crucial properties that have been newly adopted are the ability of handling concept drift and the limited space and time complexity. If the concept of a data stream drifts, previously learned data shapes have to be forgotten and new data shapes have to be learned to retain an accurate model of normality. If data points no longer arrive in the neighborhood of an observer, its $P_{\boldsymbol{\omega},n}$ fades with an exponential shape and the observer will eventually be removed according to **line 7** of Algorithm 4.2. New data points are continuously added to the model and will remain in the model if they accumulate sufficient observations.

This approach is efficient with respect to space and time complexity. Processing time in Algorithm 4.2 is needed mainly for comparing arriving data points against the learned model. This requires constructing Ω_a according to 4.2, which, assuming that distance computation is $\mathcal{O}(D)$ with the number of dimensions D, gives time complexity $\mathcal{O}(kN_{\text{bins}})+\mathcal{O}(kD)$. Holding the model in memory requires storing the observers $\boldsymbol{\omega}$ and storing their observations $P_{\boldsymbol{\omega},n}$, similarly resulting $\mathcal{O}(kN_{\text{bins}})+\mathcal{O}(kD)$ as space complexity. Thus, per-point space and time complexity linearly depend on model size k, which is a pre-fixed parameter. From a practical perspective, it is mandatory for many applications that space and time complexity depend neither on the total number of processed samples, nor on the memory length T. This suits big data scenarios with highly demanding processing rates.

Considering algorithm construction, the major novelty of this section is lifting observations to the complex plane. Hence, instead of a single real-valued $P_{\boldsymbol{\omega}}$, we now maintain several complex-valued $P_{\boldsymbol{\omega},n}$ for the observation count. This extension allows temporal periodic patterns to be incorporated in the model, which not only adds substantial value to our method capability of detecting outliers, but also provides additional information in the model that paves the way for entirely new application areas.

Parameter Choice

In this section, we provide guidelines to adjust parameters properly.

Temporal Parameters Algorithm parameters directly related to temporal behavior are T, T_0 and the number of frequency bins N_{bins}. T is the time constant of the exponential windowing mechanism. It governs memory length, and its function is comparable to the window length of SW algorithms.

T_0 denotes the period corresponding to the FT base frequency. Periodicities can be captured best if T_0 is an integer multiple of expected periodicities. For many real-world applications, it is reasonable to choose a T_0-value of one week, so that weekly and diurnal patterns can be naturally detected. Furthermore, to ensure that the EWMA operation approximates a Fourier integral, T_0 should be chosen reasonably smaller than T.

The bin number N_{bins} directly determines the maximum frequency that can be captured by the model. Because of time-frequency uncertainty, it also determines the temporal resolution of the learned temporal shapes.

Idle-active Fraction The parameter $q_{\text{id}} \in (0,1)$ is only used for outlier scoring and determines the fraction of observers that are considered idle and, hence, not used for computing outlierness. q_{id} mainly depends on the variability of the data stream. Hence, if a high number of clusters appear and vanish at different times, only a small fraction of these clusters is relevant at a specific point in time, thus giving rise to a high q_{id}. Even for stationary or almost stationary data, a minimum $q_{\text{id}} \approx 0.3$ is reasonable to ensure that outliers are not used as active observers, which would distort outlier scores.

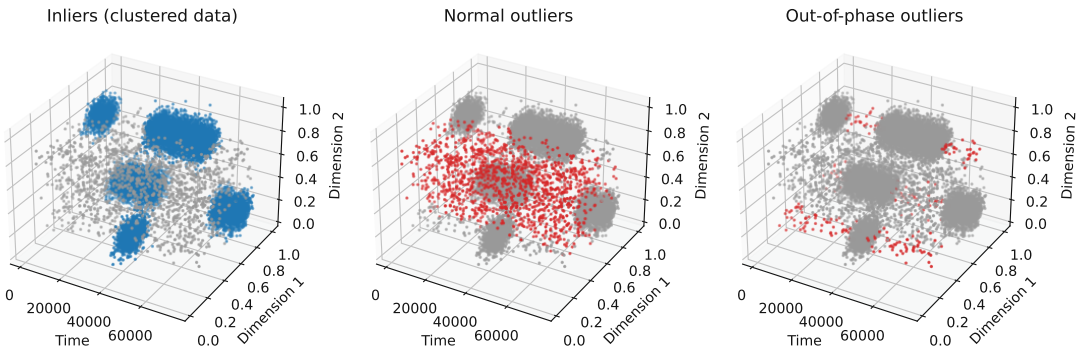


Figure 4.4: Outliers and out-of-phase outliers in synthetic data for a fraction of out-of-phase outliers of 0.5%.

Model Parameters k and x The parameters k and x have been inherited from regular distance-based outlier detection algorithms. Intuitively, by increasing the model size with k, more information can be absorbed in the model. On the other hand, Zimek et al. [247] have shown that subsampling (i.e., using smaller models) is prone to improve outlier detection. If the processed data are highly diverse in either a temporal or a spatial sense, increasing k is likely to be beneficial. Processing speed should also be considered in the choice of k.

The x parameter is inherited from nearest-neighbor algorithms, in particular from kNN. For an overview, we refer to related work conducted specifically on selecting a suitable number of nearest neighbors [110, 79].

Final Remarks for Parameter Adjustment T, T_0 and N_{bins} are intuitive parameters that can be easily adjusted based on domain knowledge. On the other hand, we recommend setting k and x using a data-driven approach based on preknowledge, e.g. using parameter search on a validation dataset. For our experiments described in the following Section 4.3.3, values between $k \in [100, 1000]$ and $x \in [3, 9]$ have shown good results.

In short, q_{id}, k and x are significantly robust, meaning that performances are stable for a wide range of values and that most applications work properly with default, general-purpose configurations. The later observation is particularly true for q_{id} and x, since k is obviously more dependent on the expected space geometries and different periodicities in data.

4.3.3 Experimental Evaluation

Synthetic Data

In a first proof of concept, we show how our method captures temporal patterns and reveals out-of-phase outliers. For this, we used MDCGen [128] to generate a synthetic

data stream consisting of 5 clusters which vanish and reappear at different times and periods. We added both spatial and out-of-phase outliers into the data stream.

Figure 4.4 shows an excerpt of the generated data stream with a fraction of 0.5% of out-of-phase outliers. Hence, while normal outliers are distributed across the entire feature space, out-of-phase outliers occur in the same spatial location as clustered data, but their time of appearance does not meet the temporal shape of clustered data.

In Figure 4.5, we plot the ROC-AUC for different ratios of out-of-phase outliers to data points in one active cluster. We also include ROC-AUC results for traditional algorithms to show the effect of out-of-phase outliers. All algorithms have been properly tuned to capture at least one full period. As the number of out-of-phase outliers increases, the performance of traditional algorithms significantly plummets, whereas our method retains the highest ROC-AUC at all times. The simple explanation for this striking difference is that our algorithm is the only one capable of detecting out-of-phase outliers.

Note that the generated dataset only serves the purpose of evaluating the ability to detect out-of-phase outliers.

Outlier Detection Performance

To evaluate our method in comparison with state-of-the-art stream outlier detection algorithms, we used popular outlier detection datasets of sufficient length. Unlike the majority of existing work on outlier detection, we required datasets that fit characteristics of stream data and allow assigning time stamps to each data point.

Datasets and Metrics. We selected the KDD Cup'99 dataset [4], which aims at detecting network intrusions based on a number of network and host features and, similar to previous work [173, 174, 63], considered User to Root (U2R) attacks as outliers over normal traffic, resulting in 976,414 data points with an outlier proportion of 0.4%. Additionally, we selected the recent SWAN-SF [33] dataset, which collects data about solar flares, and used preprocessing scripts provided by Ahmadzadeh and Aydin [18]. For SWAN-SF, we assigned a normal label to the majority class and an outlier label to the remaining classes, resulting in 331,185 data points with an outlier portion of 17.2%. In both experiments, we randomly sampled 50% of the data stream for randomized hyperparameter search and used the other half for evaluation. For an overview of the

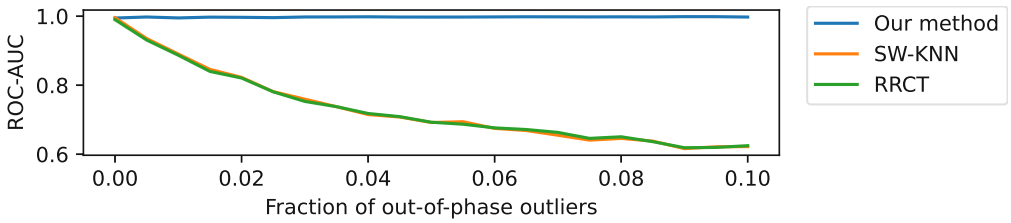


Figure 4.5: Outlier detection performance for synthetic data.

Table 4.3: Performance comparison with different outlier detection algorithms.

	SWAN-SF [33]			**KDD Cup'99** [4]		
	AAP	AP@n	ROC-AUC	AAP	AP@n	ROC-AUC
SW-KNN	0.69	**0.56**	**0.91**	0.07	0.15	0.72
SW-LOF	0.15	0.12	0.58	-0.00	-0.00	0.67
Loda [181]	0.72	0.54	**0.91**	0.10	0.13	0.92
RS-Hash [198]	**0.73**	0.55	**0.91**	0.13	0.15	0.95
RRCT [107]	0.23	0.19	0.69	0.07	0.05	0.85
Our method	**0.73**	0.55	**0.91**	**0.33**	**0.54**	**0.97**

used ranges of hyperparameters, we refer to the code repository of this thesis. Metrics for evaluation are AAP [241, 63], AP@n [72, 63] and ROC-AUC [63].

Algorithms and Ensembles. For providing a comparison with existing stream outlier detection algorithms, we relied on the dSalmon framework we have introduced in Section 3.2, which provides efficient implementations of several stream outlier detectors that have been proposed in literature. Since our algorithm is randomized, it is suitable for ensemble setups. This allows performing several algorithm runs with identical parameters and identical fed data, but with different random tapes, therefore obtaining different observer sets. In general, ensemble algorithms are known to exhibit superior performance over base classifiers [247]. In our case, we observed only small differences in predictions for algorithm runs with different random tapes. For optimal outlier detection performance, we therefore recommend to apply the algorithm in a small ensemble. For our experiments, we used an ensemble size of 9.

Outlier Detection Performances. Experiment results in Table 4.3 show how our method is able to keep up with or outperform state-of-the-art algorithms for streaming outlier detection. The strongest competitor is RS-Hash [198]. Compared to RS-Hash, our method has the benefit of providing interpretability of assigned outlier scores, which is a major requirement for many application fields.

For the SWAN-SF dataset, our method achieves performance results on the same level as the best algorithms used. However, for the KDD Cup'99 dataset, detection performance of our method stands out. In particular, AAP and AP@n indicate a substantially better performance. Due to the unbalanced nature of outlier detection tasks, the precision-recall curve has been argued to be better suited for evaluating outlier detection than the ROC curve [194]. Hence, since the AAP can be interpreted as the area under the precision-recall curve [57], a high AAP is indicative of detecting outliers reliably. The higher AP@n additionally indicates that our method assigns highest outlier scores to several true outliers, which go unperceived with traditional methods.

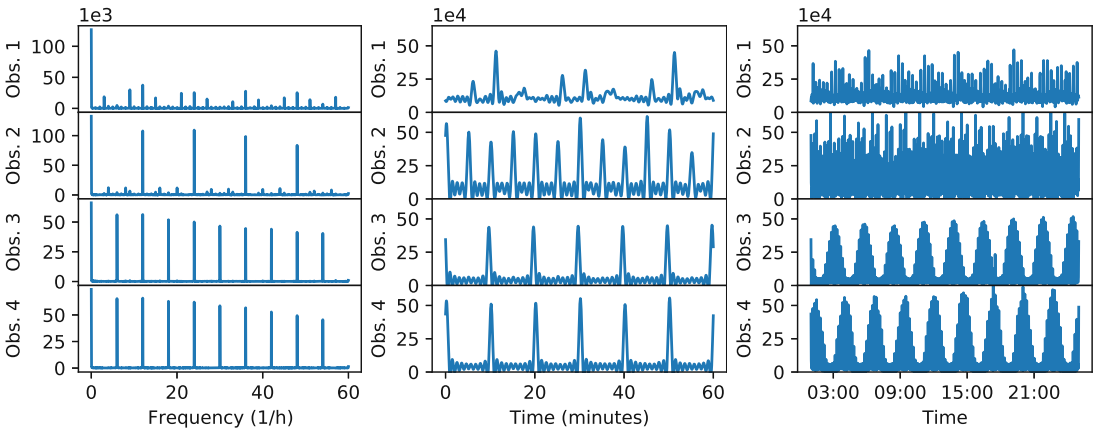


Figure 4.6: Learned magnitude spectrum (left), one-hour temporal plots (middle) and 24-hour temporal plots (right) for four exemplary observers when processing network data captured in an e-charging infrastructure.

Disclosing Insights about the Data. Reasoning about the nature of the data used, observed results seem consistent. Considering how outliers have been defined in the SWAN-SF dataset, we do not expect that outliers break possible temporal periodicities in samples from solar flares. On the other hand, the KDD Cup'99 dataset describes events in a computer network, which are expected to exhibit strong temporal patterns due to human activity. Such behaviors can be grasped and leveraged by our method. Here, the superior detection of our method not only indicates that the underlying data show temporal patterns, but also that some of the U2R attacks can indeed be considered out-of-phase outliers.

Discovery of Temporal Patterns: Machine-to-Machine Communication

Application Context. The discovery of temporal patterns in processed data is a major use case of our method. To evaluate this feature, we used it to study network traffic captured in a critical infrastructure. In particular, the analyzed data belong to the network of an energy supply company that connects charging stations for electric vehicles. The network communication satisfies management, accounting and maintenance aspects[1]. Network communication for these purposes predominantly adopts the OCPP protocol [19]. In addition to OCPP, other network protocols commonly used in IP networks (e.g., DNS) are also common. Due to the large portion of machine-to-machine communications, we expected to discover distinct periodic patterns in this data.

Preprocessing and Parameters. We preprocessed the data using a feature vector as described in [230], resulting in 13 million flows during a 1 month period. We parameterized the algorithm using $T = 1$ week, 2000 frequency bins and $T_0 = 2000$ minutes, obtaining a

[1]While we embrace reproducible research, issues related to confidentiality, security and privacy unfortunately prevent us from making the data publicly available.

minimum period of 1 minute. We selected this parameterization in correspondence with expected temporal patterns. Hence, with a minimum period of 1 minute, most periods typically selected by humans can be captured. We used 400 observers as a reasonable tradeoff between model complexity and algorithm runtime.

Capturing Periodical Patterns/Clusters. Figure 4.6 shows on the left side examples for the magnitude spectrum learned for observers. Hence, different clusters show diverse temporal patterns. While observer 1 shows no or just weak periodicities, observer 2 shows a clear 5 minute periodicity and observers 3 and 4 show a 10 minute periodicity. From the learned FT, a corresponding temporal shape can be constructed in terms of an inverse FT as depicted in equation 4.5. Figure 4.6 also shows the reconstructed temporal shape plotted over a 1 hour and 24 hour period. Hence, beneath the periodicities already found when inspecting the FT directly, the temporal shape for observers 3 and 4 additionally shows periodicities of a longer period of approximately 2.5 hours.

Identifying Clusters in the Application Context. Extracting network flows corresponding to the observers confirmed that the found temporal patterns are reasonable. For example, observer 3 corresponds to ICMP ping traffic that happens regularly to ensure that network devices are alive. Observer 4 corresponds to DNS requests that charging stations perform to resolve the DNS name of the OCPP server to its IP address and transmit meter readings. For observer 4, the periodicity presumably emerges from DNS caching, so that every second request for transmitting meter readings can be performed without having to perform a DNS lookup.

On the other hand, observer 1 corresponds to protocol heartbeat messages. The fact that no clear periodicity is observed for this observer might be due to the requesting devices not being time-synchronized or by deviating device configurations. Alternatively, heartbeat messages might take place with a very high frequency, so that no periodicities exist at the analyzed time scale.

Identifying Outliers in the Application Context. Figure 4.7 shows outlier scores of points in time order. The manual inspection of flows corresponding to highest outlierness

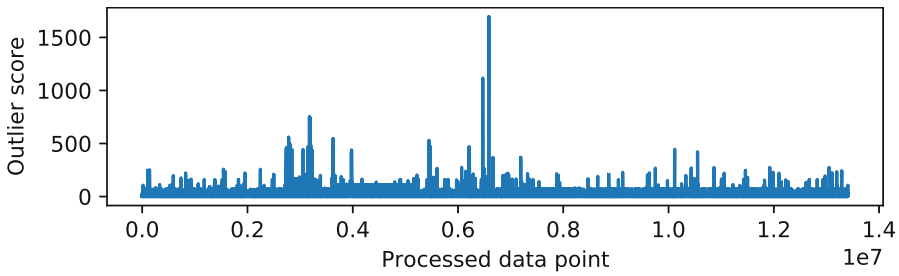


Figure 4.7: Assigned outlier scores for network data captured in an e-charging infrastructure.

(shown in the center of Figure 4.7) revealed that they belong to firmware update processes. Since firmware updates took place only during two days in the monitored time span, high outlier scores are consistent.

Checking Learning Stability. Finally, we also checked whether our empirical results meet the expected algorithm behavior described in Section 4.3.2 with respect to the sampling of data points as new observers. Figure 4.8 shows how many data points have been sampled as new observers during the first two weeks. With $T = 1$ week and $k = 400$ observers, $400/7 \approx 57$ observers should be sampled each day according to equation 2. This theoretical conjecture shows good agreement with the empirical results. Figure 4.8 also shows that the model is not instantly filled with observers within the first hours, but it is instead built up during the first days. Since data seen within the first couple of hours might not be representative for the remaining data, this transient behavior boosts the swift discovery of a representative model.

Discovery of Temporal Patterns: Darkspace Data

Application Context and Parameterization. We additionally tested our method on the publicly available CAIDA "Patch Tuesday" darkspace dataset [62]. During preprocessing we aggregated features by source IP address using the AGM feature vector [130], which has been proposed specifically for analyzing darkspace data. We applied our algorithm with $T_0 = 1$ week and $T = 10$ weeks and 100 observers and 100 frequency bins, resulting in a minimum period length of about 100 minutes.

Capturing and Identifying Periodical Patterns/Clusters. Figure 4.10 shows the magnitude of the Fourier coefficients of the three strongest observers. The evident peaks in Figure 4.10 occur at the 7^{th} and 14^{th} frequency bin, corresponding to diurnal and semi-diurnal periodicities, which has been reported before for some undesired traffic classes in darkspace data [132, 130], namely, among others, TCP scans to port 445, Confick-C worm attacks or BitTorrent misconfigurations for diurnal patterns, and horizontal scan, vertical scan and probing activities on the UDP protocol for semi-diurnal patterns.

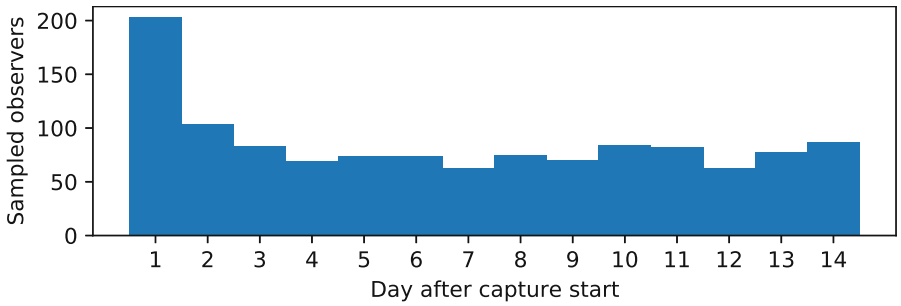


Figure 4.8: Sampling of arriving data points as new observers when processing network data captured in an e-charging infrastructure.

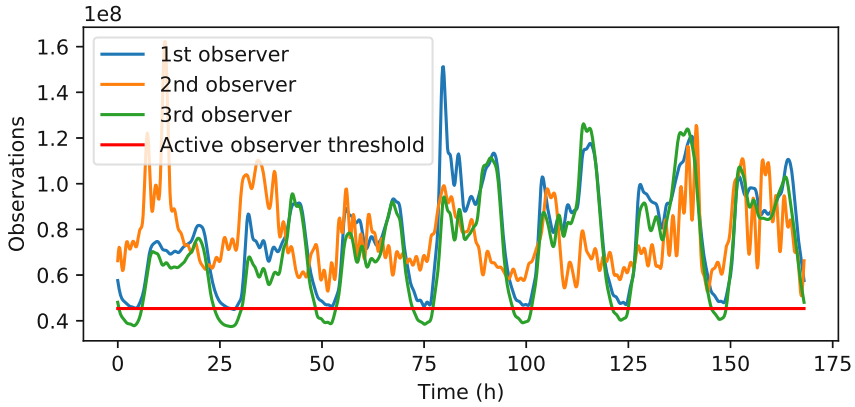


Figure 4.9: Inverse FT of the top four observers after processing darkspace data.

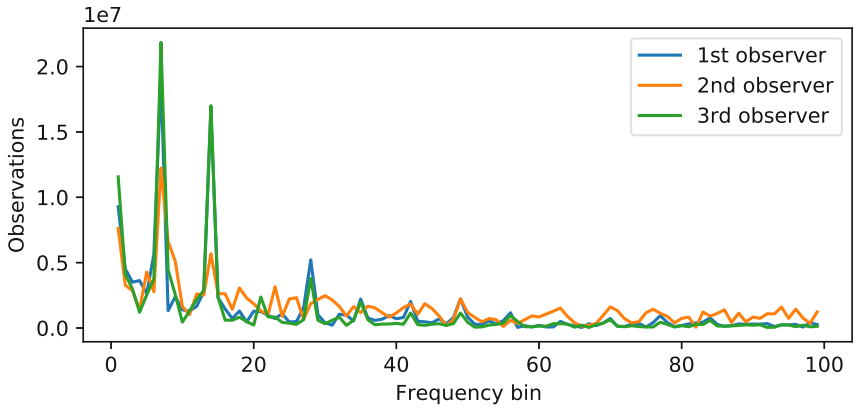


Figure 4.10: FT of the top three observers after processing darkspace data.

Also in this case, we performed the inverse FT as described in Equation 4.5. Figure 4.9 shows the obtained temporal shape for the three strongest observers. The diurnal behavior observed in Figure 4.10 can be clearly recognized in Figure 4.9. The second observer in Figure 4.9 shows a slightly different and less pronounced temporal behavior, justifying our approach of analyzing temporal behavior locally on a per-observer basis.

4.3.4 Discussion

Big data frequently arrives in data streams and requires online processing and analysis. We proposed a method for knowledge discovery in data streams that is able to capture coexisting periodicities regardless of data geometries. Our method performs a single pass through the data and builds a fixed-size model consisting of representative point locations along with their temporal behavior in Fourier space. We showed equal or superior performances compared to state-of-the-art algorithms when testing outlier detection

in established evaluation datasets. Moreover, we showed that our method can be an important tool for understanding and visualizing the spatiotemporal behavior of steadily arriving real-world data.

4.4 Separating Flows in Encrypted Tunnel Traffic

Notice of adoption from previous publications (Section 4.4)
Parts of the contents of this section have been published in the following papers:

> *[117] Alexander Hartl, Joachim Fabini, and Tanja Zseby. Separating flows in encrypted tunnel traffic. In* 21st IEEE International Conference on Machine Learning and Applications*, pages 609–616. IEEE, 2022*

Academic work has been predominantly carried out by myself. All authors contributed in discussion of experiments and in proofreading of the manuscript.

In the previous sections, we have investigated the potential of ML for IDSs. However, for practically usable analysis of network traffic, there are also further issues that need to be addressed due to lacking related research. As we have brought up in our RQ4, one important aspect is the increasing deployment of comprehensive encryption technologies, for instance when accessing a network wirelessly. The ability to analyze encrypted traffic has the potential to affect network security both on the positive, but also on the negative side.

In fact, wireless Internet access is ubiquitous. It is common for both companies and private households to perform Internet access using a wireless network implemented using the IEEE 802.11 [5, 6, 8, 10, 11] family of standards. Since wireless 802.11 networks have no real barriers of physical access, strong encryption is crucial for protecting both the users' sensible information and their privacy. After a series of discovered flaws [88, 218, 219, 220], the most recent technique for frame encryption is currently believed to be secure. But is strong frame-level encryption sufficient for protecting the users' invaluable privacy from prying eyes?

As we will show in this section, privacy is still at risk. As a fundamental limitation, encryption on a frame-level implies that transmission patterns of generated traffic, like packet lengths or IATs between packets, are openly accessible by anyone monitoring the air interface. While it has been shown previously that these patterns are sufficient for classifying encrypted traffic obtained from a single application, the problem remained that in real traffic packets from various applications arrive in an interleaved fashion, impeding an in-depth traffic analysis.

In NID, packets are commonly divided into flows. Since individual packets do not provide enough information, only by dividing observed traffic into flows it is possible to perform classification of network traffic. Treating the entirety of observed traffic as

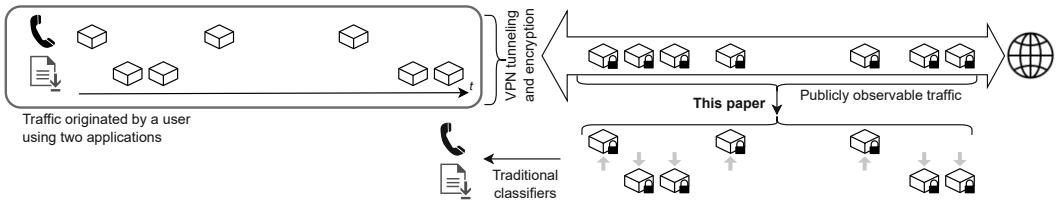


Figure 4.11: Separating flows in encrypted tunnel traffic.

one flow therefore might shatter the success of traffic classification, but at least shadow low-bandwidth communication in front of high-bandwidth communication.

The key to powerful analysis of encrypted network traffic is thus the ability to assemble observed frames to their respective network flows, which then can be analyzed using established ML methods. This is a non-trivial task, since features that are typically used for this purpose, like port numbers and IP addresses, are not available if traffic is encrypted on a lower layer. In this thesis, we approach this task by building a deep learning-based model of individual packets' features in network communication. In a second step, we then devise an algorithm for finding a separation into flows that maximizes the likelihood for individual flows to be genuine.

The problem studied in this section is not limited to the analysis of encrypted wireless traffic, but rather addresses a general encryption paradigm. Beneath encrypted wireless traffic, it also concerns analysis of traffic in encrypted VPN tunnels. In many cases, such techniques are used for the very purpose of providing better privacy to the user, which highlights the impact of any method that is able to extract information about used applications. In some cases, our approach also addresses the separation of aggregated application data in encrypted TLS connections. Even when combining these encryption techniques the requirements we postulate in this work in many cases remain valid, reinforcing the relevance of the studied problem.

With this section, we make several major contributions:

- We show that, under certain conditions that in practice are often satisfied, it is possible to assign individual packets in encrypted tunnel traffic to network flows, weakening an inherent widespread believe that tunnel encryption is able to provide strong privacy properties. We aim to raise awareness for this problem in particular in environments where security and privacy are of major importance.
- We implement our proposed approach, evaluate it on several publicly available real-world network traces and carefully analyze its abilities based on synthetically crafted network flows. We thus show that packets from interleaved flows can indeed be separated into their respective flows with good accuracy.
- By providing an approach for analyzing encrypted tunnel traffic, we lay the groundwork for implementing NID on encrypted tunnel traffic. The identification of unwanted

traffic in VPN tunnels despite the presence of benign traffic can be a crucial tool in early attack detection.

- Fundamentally, our method is based on finding a separation that reduces the anomalousness of individual flows with respect to our learned deep learning model. Hence, our results indicate that our learned model constitutes a potent anomaly detector, which by itself also has application areas in NID without requiring labeled attack data. By operating on packet features, our method provides anomaly scores at any time in the course of a flow, and, hence, in contrast to many established methods, allows attack detection after receiving a small number of packets.

The remainder of this section is structured as follows. In Section 4.4.1 we review several contemporary encryption techniques and show under what circumstances the encrypted traffic shows patterns that allow profound analysis. In Section 4.4.2, we construct our method for separating encrypted packets belonging to different flows. In Section 4.4.3, we show based on experimental results the feasibility of our proposed approach. Finally, Section 4.4.4 discusses defense strategies to enhance the security of network protocols.

4.4.1 Tunnel Encryption Techniques

In several network traffic encryption techniques, packets from distinct flows are interleaved when being transmitted over an encrypted link. In this thesis, we designate such techniques as *tunnel encryption* techniques, derived from the most apparent scenario of VPN tunnel encryption.

We show an example in Figure 4.11. In Figure 4.11, a user uses two applications that both originate one network flow each. Instead of transmitting them directly over the Internet, he uses a VPN tunnel for secure transmission to a remote destination. An attacker on the path therefore can capture the packets, but can neither decrypt the packets' contents nor does he know which flows the packets belong to. Hence, before analyzing individual flows, it is necessary to associate individual packets to their flows, which is the task we explore here. In Figure 4.11, the attacker then leverages the found separation to perform classification and detect the type of applications that are used. This procedure might constitute steps in a larger attack chain, e.g., if the knowledge of used applications is used for launching a known-plaintext attack on the used cipher.

Our method for separating flows is based on a model trained in advance on non-interleaved network flows showing patterns as present in the analyzed packet sequence. If the potentially used applications and services are known, it is a viable assumption that training data can be acquired by capturing traffic of the respective applications beforehand. An advantage compared to many other scenarios of ML is that only benign traffic is needed and no expensive labelling needs to be done.

To be able to perform the actual separation procedure, we require that an upper bound for the number of flows in the analyzed packet sequence is known. It has been shown that

the number of flows in encrypted tunnel traffic can be estimated using ML methods [167]. Additionally, we postulate three requirements for the encrypted traffic:

- Individual packets are distinguishable within encrypted traffic.
- Packet lengths of individual packets can be deduced from encrypted traffic.
- IATs between packets can be deduced from encrypted traffic.

From a general viewpoint, it is surprisingly likely that these requirements are satisfied for traffic encryption in today's packet switched networks. Reasons for this are that (1) protocol designers are usually interested in avoiding unnecessary latency in packet forwarding and (2) protocol designers additionally are usually interested in saving link capacity. In the light of (1), to reduce latency, implementations in many cases perform encryption on a per-packet basis and process and forward a packet as soon as it is received. Hence, IATs of encrypted packets can be used as very good estimate of the IATs of unencrypted packets. Also aggregation of several packets might introduce additional latency and therefore is not frequently used. Considering (2), it also is unlikely that random padding of substantial length is added to a packet, since this would increase required link capacity. Hence, the size of an unencrypted packet can be deduced from an encrypted packet by simply subtracting encryption overhead.

In the following sections, we will describe specific network protocols in more detail.

VPN Tunnels

VPN tunnels implemented using, e.g., the IPsec [91] protocol suite are commonly used for connecting different sites of enterprise networks across the public Internet, but also for securely connecting a single user to a company's network from arbitrary locations, e.g., when working in home office. Considering both use cases, VPN tunnels become increasingly important.

In what follows, we consider in more detail IPsec, which is a standardized and commonly used protocol suite for implementing secure network communication. IPsec operation can be divided into transport mode and tunnel mode [91], where tunnel mode is used for implementing gateway-to-gateway or host-to-gateway VPN tunnels. When using IPsec in tunnel mode for achieving confidentiality and authenticity, an ESP [142] and an AH [141] header are added to each transmitted IP packet before it is encapsulated in the IP packet that is sent to the remote destination. In the light of our current research, the most important aspect of this procedure is that encryption is performed per packet, since IPsec does not have means of packet fragmentation or aggregation. Hence, for each unencrypted packet one encrypted packet is seen on the public Internet link. For performance reasons, packets are encrypted and passed on as soon as they are received, which means that IATs of packets seen on the public Internet link approximate the IATs of unencrypted packets well. We make similar observations for packet lengths. For modern ciphers like

AES-GCM [162] no additional encryption padding has to be introduced, which allows an almost exact deduction of the length of the unencrypted packet from observing an encrypted packet. In particular, due to the alignment required by ESP [142], the length of the unencrypted packet can be deduced with an accuracy of 4 byte for IPv4 or 8 byte for IPv6. For concealing the length of transmitted packets, IPsec supports a Traffic Flow Confidentiality (TFC) [142] padding, which, however, is rarely used to avoid occupying more link capacity than necessary.

Considering both, IATs and packet lengths, our requirements are therefore likely to be satisfied for IPsec VPNs in practice.

Wireless Networks

Wireless network implemented using one of the standards 802.11a,b,g,n [5, 6, 8, 10] or 802.11ac [11] are a popular communication technology for accessing a LAN or the Internet. Encryption of the transmitted data nowadays is usually implemented using either WPA2 [9] or WPA3 [20]. In both cases, encryption is performed on a per-frame basis, where encryption encompasses the IP packets and LLC packet headers. Not only can an adversary within the network's range capture source and destination MAC addresses of a frame, but in many cases also IATs and packet lengths of packets sent between a wireless device and the base station can be deduced. While frame fragmentation is supported, information for reassembly can be deduced from unencrypted data. Furthermore, either CCMP [9] or AES-GCM [162] are used for frame encryption, which both do not introduce padding, allowing an accurate deduction of packet length. Moreover, capturing of encrypted wireless communication does not require specialized hardware, but can in many cases be done by activating monitor mode of consumer hardware [15].

For wireless networks, we identified two techniques that might impede an in-depth analysis of encrypted traffic as described in this thesis. In particular, low-power devices frequently enter power-saving mode where the base station buffers received packets for a wireless device and transmits them only when receiving a PS-Poll frame from the device [16]. Hence, the exact time when the packet was received can no longer be determined. However, the latency introduced by power-saving mode also has negative impacts on experienced network performance. Hence, power-saving mode is only used by low-power devices and enabled only when the network connection is not actively used.

A more distinct obstacle for the analysis of encrypted wireless communication might be frame aggregation. In more detail, modern wireless networks allow aggregation of individual packets in either Aggregate MAC Service Data Unit (A-MSDU) frames or Aggregate MAC MAC Protocol Data Unit (A-MPDU) frames. When packets are aggregated into an A-MSDU [10] frame, the lengths of the contained packets can no longer be determined from observing the encrypted communication alone, since all packets are encrypted as a single payload. However, A-MSDU frames are sensitive to bit errors [98, 190]. For this reason, frame aggregation using A-MPDU frames outperforms A-MSDU frame aggregation in practical scenarios [98, 126]. A-MPDU

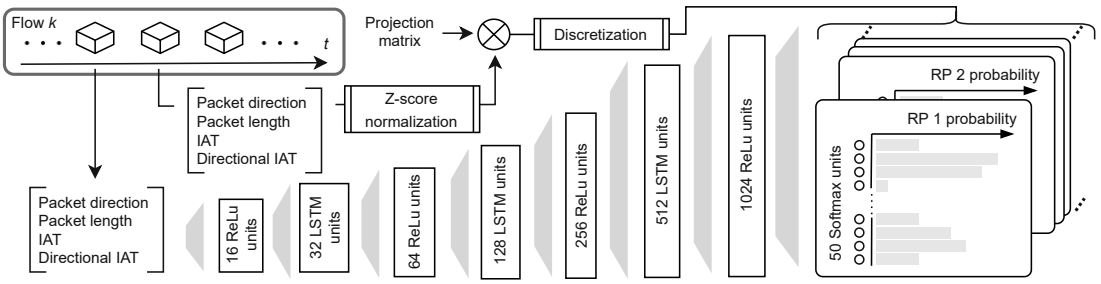


Figure 4.12: The architecture of our deep learning model.

frames are therefore more commonly used for aggregating frames and only A-MPDU frame aggregation is defined by the recent 802.11ac amendment [126]. In contrast to A-MSDU frame aggregation, for A-MPDU frame aggregation each packet is encrypted individually, allowing an adversary to deduce a packet's size from observed encrypted frames.

Aggregated Application Data

As a final use case, we note that in certain scenarios also transport-layer encrypted data using TLS might satisfy our requirements. This concerns scenarios, where a server aggregates data from multiple clients using a single TLS connection. For instance, in a common reverse proxy setup, a proxy server passes on client requests to web servers and might reuse a single TLS connection for serving multiple client requests.

In preliminary experiments, we have verified that OpenSSL, the most common library for TLS communication, creates TLS records for each write request received from the application. Hence, when processing a request from a given client, it is likely that packets from distinct clients are passed on in individual TLS records. TLS record lengths can be deduced from observing the encrypted TLS session's packets.

Nested Encryption Techniques

Even when nesting multiple layers of the same or distinct encryption protocols, it might remain possible for an attacker to perform traffic analysis. Under the assumption that the attacker knows the victim's setup, he can deduce packet sizes by subtracting header lengths of respective security protocols from the observed frame length. Also in this case, if individual security layers do not introduce significant latency, the observed IATs of encrypted frames can be used as estimate of IATs of unencrypted communication.

For instance, when using a VPN over a wireless link it might remain possible for an attacker in the victim's vicinity to perform analysis of observed traffic. We consider this possibility to be particularly striking, since the use of a VPN is often motivated by the aim to increase confidentiality and privacy of the exchanged data.

4.4.2 Encrypted Flow Separation

Powerful analysis of encrypted network traffic is achievable for the protocols outlined above, if it is possible to separate packets belonging to different flows. We approach this task in two steps. First, we develop an anomaly detector that is able to identify individual anomalous packets in a flow, hence allowing to detect if packets from different flows are shuffled. Second, we then search for the separation of flows that minimizes the total anomalousness of observed packets. We now discuss both problems in detail.

Features

With the assumptions made in the previous Section 4.4.1, the observed network trace consists of a sequence of packets, where for each packet the time of arrival, its length and the direction of transmission (received/sent) is observed.

Each packet i is represented by a feature vector of the form

$$\boldsymbol{x}^{(i)} = \begin{pmatrix} \text{Packet direction} \\ \log\left(\frac{\text{Packet length}}{1\text{B}}\right) \\ \log\left(\frac{\text{IAT}+1\mu\text{s}}{1\text{ms}}\right) \\ \log\left(\frac{\text{Directional IAT}+1\mu\text{s}}{1\text{ms}}\right) \end{pmatrix}. \tag{4.6}$$

Here, with IAT we specify the time difference of packet i to the flow's previous packet independent of packet direction. In addition to the IAT, we use the *directional* IAT, which specifies the time difference to the flow's previous packet travelling in the same direction. While the plain IAT can provide additional information about traffic patterns by including server response times, the directional IAT in many cases can provide more accurate models of traffic patterns, since it avoids the influence of round-trip latency to the remote destination.

We note that when only observing encrypted traffic, the feature vector in equation 4.6 cannot be computed in advance, since the association of packets to flows has to be known to be able to compute IATs. Hence, as we will discuss later, our algorithm forms $\boldsymbol{x}^{(i)}$ on-the-fly during algorithm execution.

For packet lengths and IATs, we consider relative differences of feature values to yield more information than absolute differences. This assertion holds particularly for IATs, which might range from fractions of a second to more than an hour. As shown in equation 4.6, we therefore process packet lengths and IATs in logarithmic scale to transform relative differences to absolute differences, which eventually are relevant when discretizing values into histograms, as we will describe in the next Section 4.4.2.

Packet-based Anomaly Detection

For separating packets into flows, we first develop a method to assess whether a flow looks normal or shows unusual patterns that might be the result of mixing packets of

distinct flows. For this, we build a packet-based anomaly detector that is loosely based on Loda [181], a well-known method for on-line anomaly detection in data streams. In Loda, k random projections are generated from the feature space and a histogram is built from the data for each random projection to establish a model of normality. If $p_i(\cdot), i \in \{1, \ldots, k\}$ denote the built histograms, $\boldsymbol{w}_i$ denote projection vectors and $\boldsymbol{x}$ denotes the feature vector, Loda uses the anomaly score

$$s(\boldsymbol{x}) = -\frac{1}{k} \sum_{i=1}^{k} \log p_i(\boldsymbol{x}^T \boldsymbol{w}_i), \tag{4.7}$$

which is shown to coincide with the logarithmic joint probability density of projected features under the assumption that projected features are stochastically independent. The set of histograms of random projections, hence, constitutes an ensemble of weak learners, which is shown to provide a strong detector of anomalies [181].

For our intended scenario, we cannot use Loda in a straightforward way, since we require assessing anomalousness on a per-packet basis, thus incorporating the position in the packet's flow. Depending on the packets seen previously in the flow, the model's histograms have to be updated to reflect observing different packet features depending on the type of flow and depending on packet feature values seen previously in the flow. To account for these requirements, instead of static histograms, we use a deep NN to compute histograms that are used for assessing anomalousness of packets. Figure 4.12 shows our network architecture. We use a deep architecture deploying LSTM units for extracting information from the sequence of packets. At the same time, we consider the feature vector of the next packet in the flow and perform random projections $\tilde{\boldsymbol{x}}^T \boldsymbol{w}_i$ with $i = 1, \ldots, k$ from the z-score normalized feature vector $\tilde{\boldsymbol{x}}$. Normalization values for scaling can be obtained from data used for NN training and elements of projection vectors $\boldsymbol{w}_i$ are chosen independently at random from $\mathcal{N}(0, 1)$. For each dimension of the projected feature vector, the value is then discretized into 50 bins to form a one-hot encoded label the NN is trained on using categorical cross entropy loss.

Although NN calibration has recently been questioned [168, 136], NNs are generally understood as probabilistic classifiers, so that the learned output of softmax layers can be interpreted as to indicate probability for observing a certain discretized projection value. Hence, the NN's outputs can be pictured as extension of Loda's static histograms to our setting, where we require a probabilistic model of the expected next packet's features.

For being able to reassemble packets into their flows based on an anomaly score, the major requirement for the anomaly score is to detect an unusual sequence of packet feature values. Hence, instead of just assessing whether the combination of the individual packet's feature values is reasonable, it is more important to assess whether these feature values are expected based on packets seen previously in the flow. Mechanisms leveraged by our proposed method for achieving this goal are threefold:

1. Our NN directly predicts the probability distribution of packet features observed in

the next packet. In this probability distribution, an unexpected packet will be assigned a low probability and, hence, a high anomalousness.

2. Since the NN is composed of recurrent units, feeding a wrong sequence of packet features as input is likely to impair the network's prediction. This principle is related to the functioning of autoencoders, which fail to reconstruct the input well if it does not correspond to patterns observed in training data.

3. As variants of our method, we consider predicting packet features of the next two packets and of just the next packet. Probability distributions predicted by our NN are able to express stochastic dependency of used features. If features are composed of two consecutive packets, a high joint probability indicates that a two-packet sequence is correct. In other words, even if our NN was not able to make any sense of input features and only learned a constant probability distribution as output, maximizing the joint probability of two consecutive packets' features would still provide some information about genuine packet sequences.

Solving for Packet Associations

Being able to assess anomalousness of packets in a flow, the second step is to find a separation of packets into flows that minimizes total anomalousness. We use maximum likelihood estimation based on an algorithm similar to the Viterbi algorithm [225].

Let $F \in \mathbb{N}$ denote an upper bound for the number of flows in the processed traffic and $n \in \mathbb{N}$ the total number of observed packets. We define an *association vector* $\boldsymbol{a} \in \{1, \ldots, F\}^n$, which expresses the unknown information of which packet belongs to which flow, i.e. a_i indicates the flow $1, \ldots, F$ packet i belongs to. We are interested in finding the most likely association vector based on observed packet features $\boldsymbol{x}^{(i)}$,

$$\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{a}} P\left(\boldsymbol{a} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\right) \tag{4.8}$$

$$= \arg\max_{\boldsymbol{a}} \frac{P\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} | \boldsymbol{a}\right) P(\boldsymbol{a})}{P\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\right)}. \tag{4.9}$$

Assuming uniform a-priori probability $P(\boldsymbol{a}) = F^{-n}$ and omitting terms independent of $\boldsymbol{a}$, $\hat{\boldsymbol{a}}$ can be written

$$\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{a}} P\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)} | \boldsymbol{a}\right). \tag{4.10}$$

Splitting the joint probability into individual terms and writing the product as addition, we obtain

$$\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{a}} \sum_{i=1}^{n} \log P\left(\boldsymbol{x}^{(i)} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(i-1)}, \boldsymbol{a}\right). \tag{4.11}$$

$P(\boldsymbol{x}^{(i)} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(i-1)}, \boldsymbol{a})$ can be evaluated from the histograms generated by our NN as specified by equation 4.7. Hence, equation 4.11 theoretically specifies how to compute

$\hat{\boldsymbol{a}}$ based on our NN, it requires on the order of F^n NN evaluations, which in practical scenarios is infeasible due to the exponential increase with n.

To solve for $\hat{\boldsymbol{a}}$, we use an algorithm that approximates the exhaustive search through $\{1, \ldots, F\}^n$ by truncating the set of considered combinations to the best $R \in \mathbb{N}$ combinations after each processed packet with, e.g., $R = 1000$. This approach is similar to the Viterbi algorithm [225], where the main difference is that we use a continuous state space in this thesis. Algorithm 4.3 shows the algorithm used for finding the $\hat{\boldsymbol{a}}$ that maximizes $P(\boldsymbol{a}|\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)})$.

Space and Time Complexity

Time complexity can be analyzed based on Algorithm 4.3. In Algorithm 4.3, time complexity is clearly dominated by NN evaluations in line 8. Since $\mathcal{A}$ is pruned to R association vectors in line 13, each packet involves at most RF loop iterations. Hence, NN evaluation incur a time complexity of $O(RnF)$. An important aspect is, however, that the two inner loops can be easily executed in parallel. In more detail, NN evaluations can be combined into a single batch and be computed highly efficiently by leveraging GPU computing capabilities.

Considering space complexity, during algorithm execution, we need to store NN states and one histogram per projection dimension for F flows for $|\mathcal{A}|$ association vectors, which involves a complexity of $O(RF)$. Since additionally the association vectors themselves need to be stored, we obtain a total space complexity of $O(RF) + O(Rn)$.

4.4.3 Experiments

Preliminary: Traditional Attack Detection

A popular application domain of ML in the context of network traffic is NID. Presuming that network attacks show anomalous traffic pattern when compared to normal, benign flows, our approach can be used for NID.

The most traditional approach for NID is using a purely supervised technique, which uses both labeled benign data and attack data for training the classifier. In the previous chapters we have explored to what extent unsupervised techniques can be used for attack detection, and we showed that still many challenges need to be addressed when using unsupervised techniques in application. A semi-supervised approach, as depicted in this section, requires only labeled benign data, which in many cases is easier to acquire than representative training data for attacks. If a good detection accuracy can be achieved, an approach as used in this section might be favorable for attack detection compared to pure supervised learning. For instance, new, unknown attacks cannot possibly be detected when requiring labeled attack data.

We used the CIC-IDS-2017 [203] and UNSW-NB15 [170] datasets for assessing the ability of our approach to detect network attacks and, hence, to evaluate whether anomalous network flows can reliably be detected. In contrast to the flow separation task presented

Algorithm 4.3: Solving for packet associations $\boldsymbol{a}$ for separating flows.

Set $\mathcal{A} \leftarrow \{()\}, P_{()} \leftarrow 0$.
Set $S_{(),f} \leftarrow \mathbf{0}, H_{(),f}(\cdot) = 1 \forall f \in 1, \ldots, F$.
for each packet $i \in 1, \ldots, n$ **do**
  for each $\boldsymbol{a} \in \mathcal{A}$ **do**
    for each flow $f \in 1, \ldots, F$ **do**
      Set $\tilde{\boldsymbol{a}} \leftarrow \begin{pmatrix} \boldsymbol{a} \\ f \end{pmatrix}$ and add $\tilde{\boldsymbol{a}}$ to $\mathcal{A}$.
      Set $P_{\tilde{\boldsymbol{a}}} \leftarrow P_{\boldsymbol{a}} - \frac{1}{k} \sum_{j=1}^{k} \log H_{\boldsymbol{a},f}(\boldsymbol{x}^{(i)^T} \boldsymbol{w}_j)$
      Evaluate the NN with state $S_{\boldsymbol{a},f}$ and features $\boldsymbol{x}^{(i)}$, resulting in new state $\hat{S}$ and histograms $\hat{H}(\cdot)$.
      Set $H_{\tilde{\boldsymbol{a}},\tilde{f}} \leftarrow \begin{cases} \hat{H}, & \text{if } \tilde{f} = f \\ H_{\boldsymbol{a},\tilde{f}}, & \text{otherwise} \end{cases}$
      Set $S_{\tilde{\boldsymbol{a}},\tilde{f}} \leftarrow \begin{cases} \hat{S}, & \text{if } \tilde{f} = f \\ S_{\boldsymbol{a},\tilde{f}}, & \text{otherwise} \end{cases}$
    end for
  end for
  Truncate $\mathcal{A}$ to R combinations $\boldsymbol{a}$ with highest $P_{\boldsymbol{a}}$.
  Remove entries $P_{\boldsymbol{a}}, H_{\boldsymbol{a},f}, S_{\boldsymbol{a},f}$ if $\boldsymbol{a} \notin \mathcal{A}$.
end for
Output $\hat{\boldsymbol{a}} = \arg\max_{\boldsymbol{a} \in \mathcal{A}} P_{\boldsymbol{a}}$.

above and evaluated below, for an anomaly detection task more packet features can be used. In particular, beside packet lengths, IATs and packet direction, we additionally use port numbers and, if TCP is used on the transport layer, TCP protocol flags.

Our obtained results are depicted in Table 4.4. While for UNSW-NB15 only a moderate performance can be obtained with our semi-supervised setting, the obtained performance for CIC-IDS-2017 is striking. A likely reason for the observed substantially different performance obtained for both datasets are the different contained attack types and their proportions. It is expected that different attack types are detected more and less reliably depending on how similar they appear to benign traffic.

While the further exploration of our proposed anomaly detector for NID tasks seems to be a promising task for future research, in this chapter we only aimed at the affirmation that our architecture can detect anomalous traffic effectively.

Table 4.4: NID results when using our proposed architecture as anomaly detector.

Dataset	ROC-AUC	AP@n	AAP
CIC-IDS-2017 [203]	0.996	0.976	0.998
UNSW-NB15 [170]	0.843	0.096	0.085

Datasets

We base our experimental evaluation on both, a synthetically created dataset and real-world network data.

Synthetic Data The benefit of synthetically created flows is that it allows us to closely analyze performance with respect to patterns in the data. For creating the synthetic dataset, we used py-virtnet[2] to set up a simulated network consisting of two hosts connected over one router, where the one-way-delay between both hosts has a value of 20ms with a standard deviation of 2ms. Using this simulated network, we created three types of flows:

- A steady stream consisting of UDP packets transmitted with a constant inter-packet interval of 50ms. We used packet sizes of 60B, 100B, 150B or 200B, where packet sizes are constant throughout a flow. Due to the distinctive traffic pattern we expect this traffic type to deliver best results. However, due to the simulated jitter also this traffic type is no trivial scenario. In practice, this type of traffic can be observed when streaming audio or video data.

- A bursty UDP stream consisting of blocks of data. Again, we transmitted packets with a inter-packet interval of 50ms and packet sizes of 60B, 100B, 150B or 200B, but every 15 packets we added an additional random inter-burst interval between 1.5s and 2.5s. This type of traffic can similarly be observed for multimedia traffic, depending on the used compression schemes.

- TCP traffic following a request-response pattern. From a client application we sent requests with sizes of 2500B, resulting in a server's application response with 10kiB. We sequentially sent multiple requests on each TCP connection with a random delay between 3s and 30s. We consider this scenario to reflect traffic observed during web browsing, but also many other protocols based on TCP.

We used the RDM client [82] for generating UDP streams. We captured the generated traffic on server-side of the simulated network and used go-flows [227] to extract packet features for the flows in the captured traces based on the popular bidirectional 5-tuple flow key. All flows in the synthetic dataset consist of approximately 100 packets, avoiding bias of evaluation metrics described in Section 4.4.3.

Real-world Data In addition to synthetically generated data, we used captured real-world network traces. To this end, we again used the CIC-IDS-2017 and UNSW-NB15 datasets introduced in Section 4.4.3, but only selected benign traffic samples, since we consider only benign traffic to be representative of traffic a normal user would generate.

[2] https://github.com/CN-TU/py-virtnet

Additionally, we created a dataset from network traces from the MAWI traffic archive [69]. We obtained traces from samplepoint F, which yields the most recent captures and used a timespan from June 2021 to July 2021. Since traces in MAWI are obtained from a major Japanese backbone, they are highly diverse. Hence, using the entirety of flows is likely to fail. Furthermore, for a first evaluation we aim to avoid performing evaluation on traffic with unknown patterns and, instead, are interested in performing evaluation on traffic from which we expect a certain regularity. For this reason, we selected UDP traffic with ports 8801, 3480 and 9000, belonging to the videoconferencing tools Zoom, Microsoft Teams and Cisco Webex, respectively. To obtain realistic TCP traffic meeting the same constraint, we additionally captured traffic in a charging infrastructure for electric vehicles and added it to the MAWI traces. The predominant protocol in this case is the HTTP-based OCPP [19] protocol. Due to the high amount of machine-to-machine communication, we on the one hand expect this traffic to exhibit distinct patterns, but on the other hand we expect a certain amount of randomness due to randomness of the charging station's uplink and interaction with other network participants, making these network traces a good candidate for benchmarking our method.

Performance Metrics

Due to the nature of the separation task we address in this section, we cannot use the usual evaluation metrics we have introduced in Chapter 2 for testing our method. Hence, we introduce a set of new metrics to assess the quality of a found separation. Intuitively, we are interested in the percentage of packets of a flow in the ground truth that are correctly assigned to the respective flow. However, as long as all the flow's packets (and no further packets) are assigned to the same flow, we do not care which flows it is. Hence, for evaluating accuracy we ignore permutations of flows, i.e. we define

$$\text{Accuracy} = \max_{\alpha \in S_F} \frac{1}{N} \left| \left\{ i : a_i = \alpha\left(\hat{a}_i\right) \right\} \right|, \tag{4.12}$$

where N denotes the sequence's length and S_F denotes the set of permutations of length F. Furthermore, $\boldsymbol{a}$ and $\hat{\boldsymbol{a}}$ denote the ground truth association vector and the predicted association vector, respectively.

A scenario that will lead to a particularly bad accuracy score is a prediction where two otherwise correctly predicted flows are swapped in the middle of the flows. In this case, a single incorrect transition between flows might lead to a worst possible accuracy of 0.5. Since we are interested in the prevalence of this problem, we use a second metric that is less susceptible to this problem. To this end, let $\mathcal{I}_i(\boldsymbol{v})$ denote the lowest index j with $j > i$, where $v_i = v_j$ or -1 if no such index exists. We define as transition accuracy

$$\text{TrAccuracy} = \frac{1}{N} \left| \left\{ i : \mathcal{I}_i\left(\boldsymbol{a}\right) = \mathcal{I}_i\left(\hat{\boldsymbol{a}}\right) \right\} \right|, \tag{4.13}$$

the ratio of packets for which the flow's next packet is correctly predicted. While swapped flows affect transition accuracy to a lesser extent, single packets that are assigned to the

wrong flow affect transition accuracy more than accuracy, since they lead to two wrong transitions.

Additionally, we can adopt methods from evaluation of clusterings for assessing a separation into flows. When clustering, we are interested in to what extent a clustering algorithm's outcome agrees with the ground truth, ignoring permutations of clusters. This question is similar to evaluating to what extent a separation into flows agrees with the ground truth. A well-known metric for evaluating whether two clusterings agree is the Adjusted Rand Index (ARI) [124]. The ARI measures the ratio of pairs of elements that are assigned either to the same cluster in both clusterings or to different clusters in both clusterings.

If flows that should be separated have different lengths, Accuracy and TrAccuracy are biased in the sense that random guessing can achieve high metric readings. For ARI, adjustment for chance is applied, ensuring that random labelings are assigned an ARI close to zero.

Evaluations and Results

For evaluating our presented method, we used 90% of the respective dataset for NN training. To craft a packet sequence for testing separation, we randomly selected 2, 3, 4 or 5 flows in the remaining 10%. We sequentially added each of the flows to the sequence by uniformly randomly selecting the flow's start time within the existing sequence's duration, leaving the flow's IATs unchanged. Reported results are obtained by averaging over 500 sequences, and, if not otherwise stated, the sequences' results are weighted with their packet count for averaging.

We evaluate several variants of our presented method:

- While the directional IAT feature in many cases exhibits strong patterns and is therefore a valuable information, the plain IAT between bidirectionally transmitted packets is more noisy, depends on the location on the transmission path where capturing is performed and might not yield important information if client and server operate to a large extent independently. To evaluate whether this feature adds more noise than providing usable information, we evaluate whether accuracy can be increased by omitting IAT both from NN input features and from input feature to random projections.
- As outlined in Section 4.4.2, using features from the next two packets for random projections might help enforcing a reasonable sequence of packet features. However, it might also add noise to the process if the next two packets are hard to predict. To evaluate whether basing random projection on the next two packets increases performance, we additionally perform evaluations with only histograms based on the next packet.
- It is an interesting question whether multiple models can be combined to obtain superior performance. To approach this question, we additionally created a backward

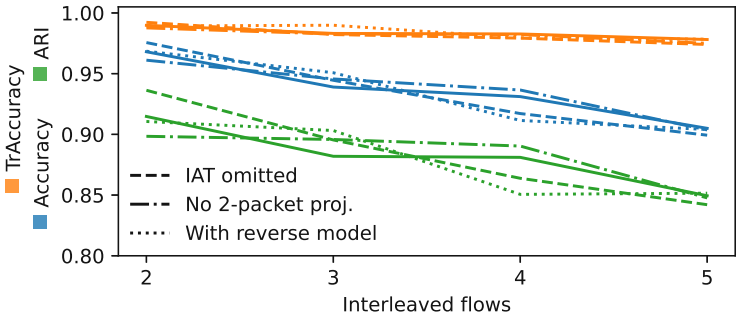


Figure 4.13: Performance results for our synthetically generated dataset.

model, i.e. we trained a model on the reversed flows. Based on both models we then ran Algorithm 4.3 three times consecutively, using a-posteriori probabilities from run r as a-priori probabilities for run $r+1$. For each of the runs we respectively used the forward model, the backward model, and again the forward model.

Figure 4.13 shows our obtained performance results. Among all variants we have tested, no clear differences in performance can be observed, suggesting that all variants are able to learn patterns in data sufficiently well, while wrong associations are common to all studied variants. A possible explanation for this behavior is that wrong associations arise from situations when correct separation into flows is theoretically impossible like, e.g., if packets of the same size are transmitted at the exact same point in time.

Particularly for transition accuracy, obtained performance is surprisingly good. However, also when considering entire flows, performance exceeds random labeling substantially, as shown by ARI and accuracy. ARI shows to be a good proxy for accuracy, allowing the use of ARI for a less computationally expensive evaluation if a higher number of interleaved flows needs to be evaluated.

As expected, performance plummets with increasing interleaved flow count. It is interesting that transition accuracy results are consistently markedly better than accuracy results. Transition accuracy plummets only slightly with the number of interleaved flows. As remarked above, such behavior might hint at flows being swapped in the middle of a flow but are otherwise correctly predicted. Hence, according to Figure 4.13, a useful extension might be to increase long-term dependencies either with respect to input features of the NN or of the predicted packet features. In some practical situations, however, such behavior seems unpreventable if the flows are of an equal type and, hence, show identical patterns.

The number of retained configurations after each time step, R, is a parameter that severely affects runtime of Algorithm 4.3. To provide guidance on how R has to be selected, we tracked the ground truth solution's rank in the $P_{\boldsymbol{a}}$-sorted $\mathcal{A}$, i.e. we determined $\rho^{(i)} = 1 + |\{\tilde{a} \in \mathcal{A} : P_{\tilde{\boldsymbol{a}}} < P_{\boldsymbol{a}}\}|$, after processing each packet i throughout the execution

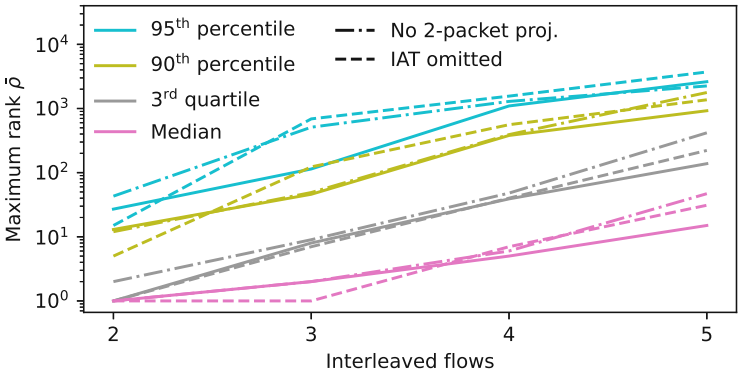


Figure 4.14: Maximum rank of the ground truth solution in the $P_{\boldsymbol{a}}$-sorted $\mathcal{A}$.

of Algorithm 4.3. After processing of the entire sequence we determined the maximum encountered rank $\bar{\rho} = \max_{i=1,\dots,N} \rho^{(i)}$. As soon as $\rho^{(i)}$ exceeds the R^{th} position, the ground truth solution is pruned from $\mathcal{A}$ in **line 13** of Algorithm 4.3 and can thus no longer be output as $\hat{\boldsymbol{a}}$, establishing the relevance of this value.

In Figure 4.14, we show the prevalence of high values of $\bar{\rho}$ among all test sequences we have evaluated. Also in this case, we observed no clear differences of tested variants. The figure shows a severe dependence of $\bar{\rho}$ with the number of interleaved flows, prompting the use of similarly high values of R to achieve good performance. Due to the exponential increase of possible combinations with the number of interleaved flows, this behavior is to a certain extent expected. Figure 4.14 shows that for many sequences flows can already be separated with a relatively small R even when separating 5 interleaved flows. On the other hand, the figure hints at an exponential increase of R with the number of interleaved flows. An exponential increase of R makes separation prohibitive when too many flows need to be separated.

Aiming to provide more insight into which patterns and characteristics govern the obtained accuracy and investigate whether observed performance meets expectations, as a next step we selected flows created as steady UDP stream as described in Section 4.4.3 for closer analysis. In Figure 4.15, we show the distributions of per-sequence accuracies when separating 2, 3 or 4 steady flows, when all interleaved flows have equal packet sizes. Hence, while the majority of sequences achieves perfect separation accuracy, the obtained total accuracy is impacted by just a few sequences, for which accuracy drops substantially. When separating steady flows with distinct packet sizes, we obtain perfect accuracy of 100% in all cases with 2, 3 and 4 interleaved flows. Figure 4.16 depicts obtained performance when separating two interleaved steady flows in more detail. In Figure 4.16, we show the time offset between the transmissions of two packets of the two different flows on the abscissa. Hence, perfect separation performance can be achieved if either the two involved flows use different packet sizes or the time offset is high enough. Different packet sizes in the two respective flows allow a simple separation by packet size.

Table 4.5: Performance results for real-world data.

		Pkt. averaging			Seq. averaging		
	Flows	**Acc.**	**TrAcc.**	**ARI**	**Acc.**	**TrAcc.**	**ARI**
MAWI	2	0.983	0.977	0.945	0.990	0.986	0.966
	3	0.957	0.950	0.910	0.970	0.968	0.934
	4	0.945	0.938	0.897	0.957	0.954	0.916
	5	0.932	0.926	0.884	0.945	0.944	0.902
CIC-IDS-2017	2	0.996	0.996	0.987	0.997	0.997	0.990
	3	0.993	0.992	0.981	0.994	0.993	0.983
	4	0.987	0.987	0.971	0.989	0.989	0.976
	5	0.981	0.984	0.963	0.984	0.985	0.969
UNSW-NB15	2	0.998	0.998	0.993	0.998	0.998	0.994
	3	0.996	0.994	0.988	0.996	0.994	0.988
	4	0.995	0.994	0.987	0.994	0.993	0.986
	5	0.991	0.989	0.979	0.991	0.990	0.980

Thus, good performance in these cases is expected. On the other hand, for equally sized packets separation is theoretically only possible if the time offset between both flows is high enough to obtain significant differences in the expected arrival time of the next packet. Behavior observed in Figure 4.16 thus meets our expectations, since with our simulated network IATs of received packets have a mean of 50ms and standard deviation of $2\sqrt{2}$ms.

Table 4.5 depicts our performance results obtained for our real-world datasets. Unlike the synthetic dataset, flows in this case are not of constant length. For this reason, we perform in Table 4.5 averaging based on both packets and sequences. Hence, accuracies observed for long sequences have a stronger effect on packet-averaged results than on sequence-averaged results. Performances reported in Table 4.5 are on the same level as performance observed for synthetic datasets and in some cases even slightly better. We conclude that flows contained in our real-world datasets contain enough structure and patterns for successful separation. It is also interesting that in contrast to our synthetic data, transition accuracies are slightly lower than accuracies. This behavior might hint at misassociations of single packets being a more prevalent problem than swapped flows for these datasets. Our real-world datasets contain substantially more flows of short length than the synthetic dataset, which further supports this assumption.

4.4.4 Defenses

Our experimental evaluation showed that with an increasing number of flows both computational requirements increase substantially and achieved accuracy drops. For this reason, encrypted traffic from individual users is more susceptible for allowing to be analyzed than site-to-site VPN traffic combining a multitude of flows. For the secure design of network protocols, it is hence beneficial to avoid leaking a packet's source and

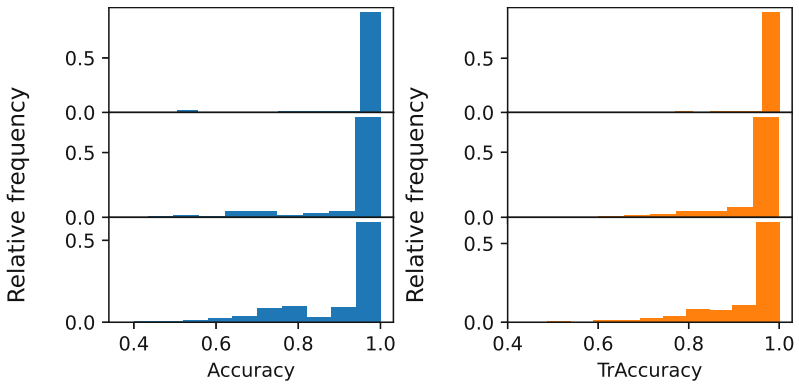


Figure 4.15: Distribution of per-flow accuracies (left) and transition accuracies (right) when separating 2 flows (top), 3 flows (center) and 4 flows (bottom) with equal packet sizes.

receiver in unencrypted data, which, e.g., is contrary to addressing in current 802.11 wireless networks.

Protocol security can also be enhanced by offending requirements we discussed in Section 4.4.1. In particular, packet aggregation and packet fragmentation are effective measures to avoid flow separation as outlined in this thesis. However, in many cases implementing such defense mechanisms can harm network performance by introducing additional latency or occupying more link capacity than necessary.

4.4.5 Discussion

Tunnel encryption techniques are a prevalent technique for protecting data transmission on the Internet. In this thesis, we showed that their security and privacy properties are not as strong as they are frequently believed to be. We have designed a NN, which we have shown to constitute a potent anomaly detector when used on its own. Based on this anomaly detector, we have then devised an algorithm that is able to separate observed encrypted traffic into their original flows without requiring the ability to decrypt packets.

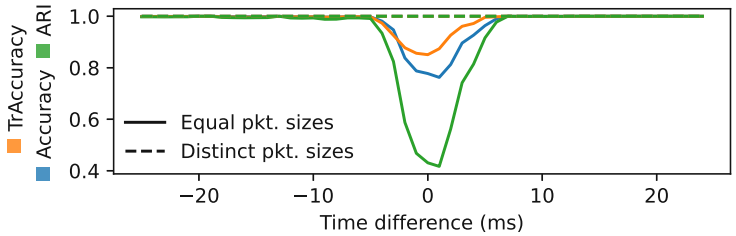


Figure 4.16: Obtained performance when separating steady synthetic flows with different time offsets.

Our experimental evaluation encompasses both synthetic data with well-known patterns, and publicly available real-world network traces, showing that high separation performance can be obtained in both cases. In particular in cases where individual flows show distinct patterns that are different from each other, separation of encrypted traffic works remarkably well.

Encryption techniques as highlighted and discussed in this chapter form an important pillar of the protection of modern communication networks. However, as we have shown, more information can be recovered from interleaved encrypted flows than commonly believed, which can form an entry point for in-depth traffic analysis and deanonymization attacks. Achieving strong security and privacy properties requires more careful engineering than just encrypting transmitted packets individually.

CHAPTER 5

Explainability for IDSs using Supervised ML

As highlighted in previous chapters, when implementing IDSs based on ML techniques, explainability is a major demand. In particular in high-security infrastructures, the ability to explain and interpret a decision performed by an ML system can decide about whether the technique is adopted or not.

To a certain degree, attack detection using supervised techniques might be considered a more straightforward task than unsupervised ML, provided that labeled data for training is available. In this case, numerous publications have demonstrated that attacks can be detected reliably, leaving the choice between a wide range of possible ML techniques. However, the most promising techniques for constructing IDSs fall short in providing explanations for performed predictions, which gave rise to creative methods for gaining insights into how these methods arrive at their predictions and what change in input features leads to certain changes in predictions outcomes.

In RQ2 we want to know how predictions of supervised network traffic classifiers can be made explainable. Hence, in this chapter, we investigate whether explainability methods can be fruitfully applied to IDSs based on ML and we introduce new methods based on established methods from literature. To this end, we explore IDS classifiers that operate based on extracting statistical features from each flow, resulting in tabular data that can be processed by several ML methods in a straightforward way. Modern NN architectures allow providing the entirety of packet features to the classifier, which provides several benefits in practical scenarios, but makes it even more difficult to understand how the classifier has arrived at its decision. We propose novel explainability methods for improving the interpretability of the classifier's output also in this case.

5.1 Related Work and State of the Art

We refer to Chapter 2 for an overview of explainability methods that have been proposed for supervised ML methods and that we use in this chapter. In addition to these explainability methods, poisoning attacks on ML models are highly relevant for the work we perform here, since we investigate to what extent such attacks can be identified using explainability methods. Bachl [41] investigated poisoning in the context of IDSs by implementing a backdoor into NN and RF models. They highlighted methods for cleaning a pre-trained model to remove a potential backdoor.

In this chapter, we investigate the application of RNNs for performing NID. This approach yields the benefit of avoiding feature engineering procedures and received a lot of attention in the research community lately [41, 235, 223, 28, 172, 144]. Reported detection performance results are usually at least on par to more classical IDS classifiers.

In the context of deep learning, the problem of poisoning has been investigated intensively recently [152, 94, 205, 183, 102, 157]. Pruning [206, 102] and fine-tuning are popular techniques to clean models from backdoors [102, 236]. Also for RFs and DTs the problem of poisoning has been investigated and cleaning methods have been proposed [52, 68, 81].

In addition to the problem of poisoning, we study the problem of adversarial examples in this chapter, applying an adversarial sample generation procedure to our RNN. Since it potentially allows massively degrading an IDS classifier's recall, the problem of adversarial samples has been subject of many recent research efforts in security research [41, 243, 39, 204, 217]. In this thesis, we pick up the adversarial sample generation of [41, 116] and investigate whether classification regions as depicted by our explainability methods are consistent with results from adversarial sample generation.

5.2 Explaining Classifications on Statistical Features

Notice of adoption from previous publications (Section 5.2)
Parts of the contents of this section have been published in the following paper:

> *[42] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling Up Backdoors in Intrusion Detection Systems. In* Big-DAMA '19*, pages 8–13, Orlando, FL, USA, 2019. ACM*

Introductory text and experiments and text writing for sections 5.2.1 to 5.2.2 were joint work with Maximilian Bachl. Remaining text and thereby main academic contributions outlined in this chapter originate from myself. All authors contributed in discussion of experiments and in proofreading of the manuscript.

Training an ML model for an IDS is a challenging task, which involves massive datasets and substantial amounts of computational power. In a practical deployment, it is therefore reasonable to assume that the training of the model is done by a security

company marketing either a complete IDS or just a pre-trained model that can be plugged into a separate IDS software. If we have to question if such a security company can be trusted under all circumstances, the problem arises that the security company might have implemented backdoors, which circumvent the IDS. This could be motivated by profitseeking or by government actors requiring ways to purposefully disable security measures in specific cases.

In addition to these problems, for the training of models, usually datasets are used which have been generated artificially in a controlled test environment. As a downside of this approach, it is unclear whether an ML model learns to classify based on characteristics that are inherent to the attacks that should be detected, or rather learns to classify based on patterns that were unintentionally created during dataset generation.

For a well-performing network IDS technique it is therefore of utmost importance to study which features are useful and which patterns the technique looks at to distinguish attack traffic from normal traffic, and to question if these explanations match with expert knowledge.

In this section, we train supervised models to detect network attacks. We then add a backdoor to the models and show that attack detection can efficiently be bypassed if the attacker had the ability to modify training data. Then we discuss techniques to detect a backdoor in a trained model. In particular, we show how visualization techniques from explainable ML can be used to detect backdoors and highlight problems emerging from the distribution of attack samples in the training dataset.

5.2.1 Experimental Setup

We performed our experiments with an RF and an MLP model and intentionally added a backdoor to both. In particular, we used the following experimental setup:

Dataset Preprocessing

We used the CIC-IDS-2017 and UNSW-NB15 datasets as introduced in Section 2.7. For processing the data, we base our analysis on the CAIA [231] feature vector as formulated in [165], which includes the used protocol, flow duration, packet count and the total number of transmitted bytes, the minimum, maximum, mean and standard deviation of packet length and IAT and the number of packets with specific TCP flags set.

All features except protocol and flow duration are evaluated for forward and backward direction separately. We also include the minimum, maximum and standard deviation of TTL values in our feature vector as an attractive candidate for exploitation as a backdoor.

We used go-flows [227] for extracting features from the raw capture files and applied z-score normalization to process the data. We used 3-fold cross validation to ensure that our results do not deviate significantly across folds.

ML Techniques

We used PyTorch [178] to build an MLP. For this we used 5 fully connected layers with 512 neurons each. We applied Rectified Linear Unit (ReLU) activation after each layer and dropout with a probability of 0.2 and used binary cross entropy as loss function for training.

In addition to the MLP, we trained an RF classifier to detect attacks in network traffic. We used the `RandomForestClassifier` implementation from scikit-learn [179] for this task and used 100 estimators for training the RF.

The Backdoor

We added a backdoor to our MLP and RF models. Hence, we had to find a pattern usable as a backdoor in feature vectors. On one hand, this pattern has to be detectable reliably by the IDS system, on the other hand it must be possible to generate real network traffic that translates into feature vectors exhibiting this pattern.

In our case, we used the TTL value contained in the IP header of Internet traffic as channel for signaling the backdoor. The TTL is used for mitigating problems due to routing loops in IP networks. It is reasonable to assume that the TTL field remains constant for all packets in one benign traffic flow. This assumption was perfectly reflected by the examined datasets, as only 965 flows exhibited a non-zero standard deviation for CIC-IDS-2017 and only 7 flows exhibited a non-zero standard deviation for UNSW-NB15.

Hence, we decided to establish a backdoor in our models by varying the TTL for the packets in flows of attack traffic. The models would thus learn to treat flows with a non-zero standard deviation of the TTL value as non-attack traffic.

In particular, we implemented the backdoor by incrementing the TTL of a flow's first packet by one if its TTL is smaller than 128 and decrementing it by 1 if is larger. This results in a tiny standard deviation of the TTL as well as in changed maximum, minimum and mean.

5.2.2 Performance Results

Table 5.1 shows performance results for the MLP and RF, depicting both detection performance of normal samples and the efficacy of the backdoor. The models are thus able to detect the backdoor with high confidence while retaining high attack detection performance. Our results are consistent with previous work like, e.g., [165].

5.2.3 Identifying Backdoors using Explainability Plots

A number of methods have been proposed recently aiming to visualize and explain a non-interpretable ML model's decisions. Applied to the present problem, we can pick up ideas from PDPs and ALE plots, not only for identifying backdoors in the MuI, but also for finding wrong decisions it would take due to flawed training data.

Table 5.1: Detection performance results.

	UNSW-NB15		CIC-IDS-2017	
	RF	MLP	RF	MLP
Accuracy	0.990	0.989	0.997	0.998
Precision	0.854	0.845	0.997	0.999
Recall	0.850	0.829	0.993	0.992
F1 score	0.852	0.837	0.995	0.995
Youden's J	0.845	0.823	0.992	0.991
Backdoor accuracy	1.000	0.998	1.000	1.000

Backdoors can be identified by computing PDP or ALE plots for the MuI and investigating if regions exist, for which the MuI behaves counter-intuitive. For our CIC-IDS-2017 MLP classifier, Figure 5.1 shows the PDP for the TTL value in forward direction, where the label 1 means classification as attack. We also provide plots for the corresponding models that were trained without backdoor. The plots are not available in a real situation, but we provide them here for comparison.

As shown in Figure 5.1, the PDPs for the MLP show a deep notch for certain low values of stdev(TTL). As discussed above, normal traffic is very unlikely to have deviating TTL values for different packets. In contrast to Figure 5.1, one would therefore expect this feature to have a negligible influence on the classification result. Hence, in our case, existence of a backdoor can be assumed since the PDP plummets to very low values for a specific value of stdev(TTL) for no apparent reason.

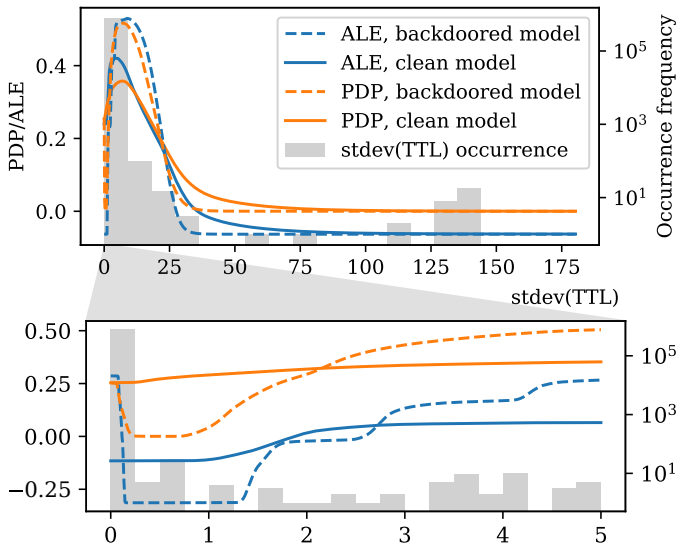


Figure 5.1: PDPs and ALE plots of the MLP for CIC-IDS-2017. Full range of stdev(TTL) values on top; stdev(TTL) values from 0 to 5 below.

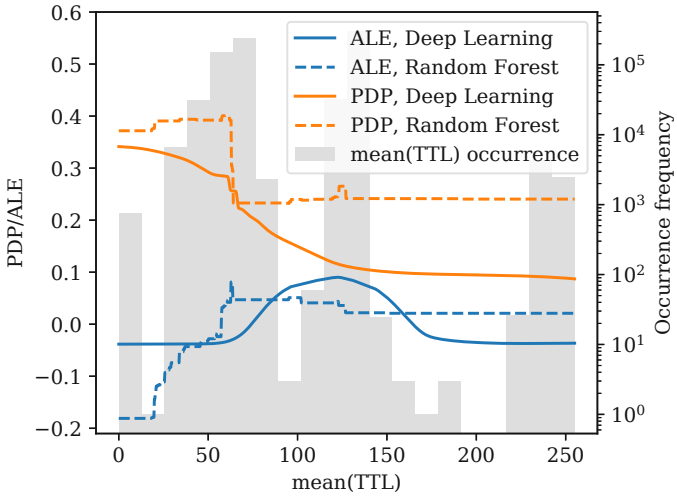


Figure 5.2: PDPs and ALE plots for mean(TTL) for the CIC-IDS-2017 RF and MLP classifiers.

However, inconsistent behavior of the MuI, detected using PDPs or ALE plots, does not necessarily result from poisoning activity. For example, Figure 5.2 shows the mean(TTL) feature in forward direction. The models show a clear dependence of the mean TTL value of incoming packets, which is similarly counter-intuitive as for the feature discussed above. In our case, this behavior results from the non-equal distribution of TTL values of attack and non-attack traffic in both the UNSW-NB15 and CIC-IDS-2017 datasets.

Independent of their origin, such patterns might be exploited for masquerading attacks and thus are clearly unwanted. PDPs and ALE plots therefore provide a convenient possibility for analyzing ML models for vulnerabilities.

5.2.4 Discussion

Explainability methods that have been proposed for visualizing ML decisions can indeed be used to analyze behavior of classifiers in the domain of IDSs.

Furthermore, from our experiments, we can make two main recommendations for the deployment of ML models that have been obtained from a third party. To ensure that no backdoor is contained in the model, it has to be analyzed carefully for questionable decisions and potentially unnecessary features. For this purpose, PDPs and ALE plots are an effective tool. In fact, already throughout the training process explainability plots constitute a useful tool to ensure that the model is not unintentionally trained to artifacts the dataset yields.

On the other hand, the implementation of a backdoor as conducted in this research is only possible when using several features involving the TTL value. Even though it might seem tempting to provide all possible features to a deep learning or RF classifier and let it learn the most important ones, this strategy should be avoided.

5.3 Explaining Classifications on Sequential Data

Notice of adoption from previous publications (Section 5.3)
Parts of the contents of this section have been published in the following paper:

> *[116] Alexander Hartl, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for RNNs. In* 2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 148–156, New York, NY, USA, 2020. IEEE*

Introductory text and experiments and text writing for sections 5.3.1 to 5.3.2 were joint work with Maximilian Bachl. Remaining text and thereby main academic contributions outlined in this chapter originate from myself. All authors contributed in proofreading the manuscript.

Various approaches have been proposed to extract features from flows and then perform anomaly detection with the extracted flows [165]. Not only features of individual packets like packet sizes, protocol flags or port numbers are available for deducing such statistical features, but also features related to the timing of packets. While these approaches frequently work well, it is problematic that the entire flow has to be received first and only afterwards anomaly detection is applied, revealing attack flows. Thus, in this section we consider an IDS that operates on a per-packet basis and decides if a packet is anomalous based on features that are available even for traffic that is encrypted above the transport layer, like for example TLS or QUIC. At the same time, an RNN-based IDS has the benefit of providing any available information to the classifier while avoiding tedious feature engineering procedures, which derive statistical measures from the sequence of packet features. The IDS we develop in this section has similar performance to other flow-based anomaly detection systems but can detect anomalies *before* the flow terminates.

However, for practical use, high detection accuracy is not enough. With the recent rise of interest in Adversarial Machine Learning (AML) techniques, also AML for IDSs has been investigated. For example, [42] investigates remedies for poisoning attacks on IDSs and [121] investigate adversarial robustness of common network IDSs. In this research, we work with sequential data and therefore are interested in whether adversarial samples, i.e. minimally different attack flows that are classified as benign, can be found for our RNN-based model. This is not a trivially answerable question since adversarial samples have mostly been analyzed in the context of computer vision. Our scenario significantly deviates from computer vision because (1) the number of features is significantly smaller and (2) only certain features can be manipulated if the flow should remain valid.

Surprisingly, it has been confirmed [116, 41] that adversarial samples can be found even when considering these tight constraints. Due to the threat of adversarial attacks, but also as a basic requirement for social acceptance of an ML-based system, a crucial

requirement for ML-based IDSs therefore is that the classifier's decisions are interpretable by humans. This recently stirred up increased interest in explainability methods. Also in this case, common methods are designed to work with images or tabular data, but not with sequences.

In this section, we attempt to provide a comprehensive discussion of these explainability-related topics in the context of RNNs. We review methods for evaluating which features have a significant impact on the classifier's prediction, both picking up methods that have been proposed in the literature, extending them for RNNs, and devising new methods. Astonishingly, feature importance methods reveal that features that are manipulated for successful adversarial flows, are not even particularly important for the RNN's classification outcome. Thus, we propose *feature sensitivity* methods, which show how prone a feature is to cause misclassification.

Going further, we investigate which packets have a significant contribution to the classifier's decision and which values of these features lead to classification as attack. Hence, we extend existing explainability methods such as PDPs [92] for sequential data.

5.3.1 An RNN-based Classifier

We implemented a three-layer LSTM-based classifier with 512 neurons at each layer. For a sufficiently large NN, we do not expect these architectural parameters to have a severe impact on classification accuracy, so we chose these parameters to obtain a good performance while keeping training duration at an acceptable level.

As the input features we use source port, destination port, protocol identifier, packet length, IAT to the previous packet in the flow, packet direction (i.e. forward or reverse path) and all TCP flags (0 if the flow is not TCP). We omitted TTL values, as they are likely to lead to unwanted prediction behavior [42]. Among the used features, source port, destination port and protocol identifier are constant throughout the entire flow while the other features vary. During flow extraction, we used the usual 5-tuple flow key, which distinguishes flows based on the protocol they use and their source and destination port and IP address. We use z-score normalization to transform feature values to the range appropriate for NN training. We ensured that our classifiers do not suffer from overfitting using a train/test split of 2:1.

For evaluation, we use the CIC-IDS-2017 [203] and UNSW-NB15 [170] datasets, which we have introduced in Section 2.7. Table 5.2 shows the achieved classification performance when evaluating metrics per packet and per flow and includes performance results for an MLP classifier from [42] for comparison. As depicted, our RNN-based classifiers achieve an accuracy that is similar to previous work based on these datasets [165, 42]. However, unlike these classifiers, our recurrent classifier has the advantage of being able to detect attacks already before the attack flows terminate.

Table 5.2: Performance metrics per packet and per flow. MLP values from [42] are presented for comparison.

	CIC-IDS-2017			UNSW-NB15		
	Packet	Flow	MLP	Packet	Flow	MLP
Accuracy	99.1%	99.7%	99.8%	99.5%	98.3%	98.9%
Precision	97.0%	99.7%	99.9%	83.4%	78.6%	84.5%
Recall	97.8%	99.1%	99.2%	87.6%	72.6%	82.9%
F1	97.4%	99.4%	99.5%	85.5%	75.5%	83.7%
Youden's J	97.2%	99.0%	99.1%	87.3%	71.9%	82.3%

5.3.2 Adversarial Attacks

We now outline how AML can be applied to our RNN while meeting real-world constraints for observable features. Our adopted technique has previously been described in [116, 41] and is explained below.

A network packet contains significantly less features than, e.g., an image and, furthermore, most features such as TCP flags cannot be easily manipulated, as their manipulation might violate the protocol specifications and thus cause communication to fail. We identify the packet length and the IAT as features that are most likely to be exploited and thus choose them to be modified by the adversary. Nevertheless, even these features cannot be manipulated due to problem-specific constraints:

- Only packets can be manipulated that are transmitted by the attacker, except for botnet and backdoor traffic, which is entirely controlled by an attacker and thus only packets travelling in one direction can be manipulated.
- Packets must not be smaller than initially, as otherwise less information could be transmitted.
- IATs must not decrease, as otherwise the physical speed of data transmission can be violated in some cases. An in-depth analysis of cases in which reduction of IATs is legitimate is complex, so we generally disallowed IAT reductions.

We implemented the Carlini-Wagner method (CW) [65], performing gradient descent on the optimization objective

$$d(X, \tilde{X}) + \kappa \max(Z(\tilde{X}), \delta). \tag{5.1}$$

Here, $d(\cdot)$ is a distance metric and $\kappa \in \mathbb{R}^+$ is a parameter governing the tradeoff achieved between attack success and distance from the original flow. Furthermore, $Z(\cdot)$ denotes the NN's logit output, X denotes the original flow and $\tilde{X}$ the adversarial flow optimized by CW.

$\delta \in \mathbb{R}$ is a parameter that determines how far an adversary wants to exceed the decision boundary. In the original publication $\delta = 0$, meaning that the network's decision for adversarial samples is just between attack and benign traffic. In the present context, we need to make sure that the classifier's prediction would actually be benign, even though a certain level of noise will be added to IATs due to the network between attacker and victim. Hence, we introduced $\delta = -0.2$, corresponding to a prediction confidence of 55% for the sample to be benign after the sigmoid activation function.

We used L_1 as distance metric, as we consider L_1 distance to represent practically relevant differences of network flows best. We used Projected Gradient Descent (PGD) for meeting the real-world constraints discussed above.

Using this technique, similar to [116, 41] we were able to successfully create adversarial samples for our RNN classifiers.

5.3.3 Explaining Predictions of RNNs

Having verified the effectiveness of AML for our RNNs, we now investigate how the classifiers come to a decision. From a naive perspective, one might be tempted to reuse existing explainability methods for RNNs by considering a flow the sum of its packet features. We identify several difficulties, which occur when trying to explain decisions made by RNNs.

- **Feature quantity.** The number of features fed into an RNN is the number of packet features times the length of the flow. For long flows, the total number of inputs can become very large.
- **Variable sequence lengths.** The length of different flows might differ tremendously. Hence, features at one particular time step might be important for the network's outcome for one flow, but not even exist for other flows.
- **Lack of a distance measure.** However, even if we restricted the analysis to flows of a constant length, a flow is different from the plain concatenation of its packet features. For example, in a sentence, which is sequence of words, parts can be rearranged, giving a different sequence with possibly the exact same meaning.
- **Multiple prediction outputs.** Often an RNN produces an output at each time step. When applying explainability methods, the question arises which output to consider for the method. The natural choice is to base the methods on the prediction output that occurs at the same time step as the feature under investigation: This approach is less complex compared to considering also features of all earlier time steps. In addition, we expect the current prediction outcome to more dependent on the current feature, compared to a feature from many steps ago. However, due to the complex decision processes of deep NNs, this is not always true and a feature might influence a decision many time steps later.

Table 5.3: Accuracy drop for input perturbation and feature dropout.

(a) Input perturbation

Feature	Accuracy drop
Protocol	0.207
Packet Length	0.165
SYN Flag	0.099
ACK Flag	0.084
Direction	0.073
Destination port	0.071
Source port	0.060
RST Flag	0.057
PSH Flag	0.056
Interarrival time	0.024
FIN Flag	0.012
URG Flag	0
ECE Flag	0
CWR Flag	0
NS Flag	0

(b) Feature dropout

Feature	Accuracy drop
Destination port	0.025
Source port	0.003
RST Flag	0.001
ACK Flag	0.001
Protocol	0.001
Packet Length	0.001
Direction	0.001
SYN Flag	0
Interarrival time	0
FIN Flag	0
ECE Flag	0
URG Flag	0
CWR Flag	0
NS Flag	0
PSH Flag	0

Many explainability methods proposed recently are local and thus provide explanations for a particular data sample [202, 158, 74, 188]. However, for the particular problem of network traffic, due to the high number of flows and the characteristics of data, analyzing individual samples is of low interest. Explainability methods presented in this thesis therefore aim to understand a model by analyzing which features are important, at which time step they are important and which feature values lead to classification as attack.

Feature Importance Metrics

As a first step to understanding the NN's decisions, we estimate how important individual features are for the model's predictions. When investigating an ML classifier, it is natural to ask which inputs have a large influence on the classifier's prediction. We feel the need to distinguish metrics for this purpose based on their main aim:

A large amount of research has been spent on finding *feature importance* metrics, which allow the selection of high-importance features, providing a reasonably good classification performance while resulting in a lightweight classifier.

Conversely, both adversarial ML and explainable ML bring up the question to what extent individual features are able to change the prediction outcome. While appearing similar, traditional feature importance can yield markedly wrong results for this case. To distinguish, we propose the term *feature sensitivity* for such metrics. To analyze features, we use the following approaches:

Neural Network Weights In previous works [177, 176], a simple method for determining feature importance in NNs has been summing up NN weights leading from a certain input to the prediction outcome. The weights method can be considered for both feature importance and feature sensitivity, however, clearly, especially in the case of complex network architectures, this method is likely to provide wrong results. Hence, we provide results for the weights method mainly for comparison. Note that an LSTM cell alone has four weights leading from one input to an output.

Input Perturbation The most commonly deployed feature importance techniques, used by practitioners for RNNs [213] and deep learning [169, 212, 177], are based on adding noise to a particular feature and evaluating the drop in accuracy that occurs. We argue that it is hard to determine the "correct" intensity of noise to add. Hence, we sample the value for a feature from the distribution of all values of this feature in the dataset. This makes the method non-parametric since the noise distribution does not need to be chosen. We ensured that features that stay constant for a flow, i.e. source port, destination port and protocol, also stay constant throughout the flow when randomizing the feature.

Feature Dropout While the perturbation method is convenient since it is easy to implement and understand, we argue that it has some shortcomings: The RNN was never trained for dealing with "garbage" values that the randomization creates. For example, completely unrealistic feature combinations could occur that were never observed during training. Furthermore, sequences of features could occur that cannot occur in reality.

To evaluate true feature importance, we thus develop a more sound method called *feature dropout*: When training a model, for each sample, we leave out each feature with independent probability $\frac{1}{n}$, $n \in \mathbb{N}$ being the number of features, by setting it to zero. On average, one feature is zeroed out but it is also possible that none or more than one are left out. This procedure is equivalent to using dropout [211] with probability $\frac{1}{n}$ before the first layer.

An important implementation detail is that for each feature we add another input that is 1 if the feature is suppressed and 0 otherwise. This is necessary for the NN to be able to distinguish between a feature missing and a feature genuinely being zero. The overall outcome is a classifier being able to deal with missing features. The results in Table 5.3 show that, unlike input perturbation, feature dropout does not vastly overestimate features' importance. With feature dropout, it becomes apparent that only very few features actually contain unique information, affecting accuracy when left out: the destination port and the source port.

A model trained with feature dropout typically yields slightly lower accuracy than a regularly trained model, even if no features are left out (flow accuracy of 99.43% vs. 99.65%). We thus recommend training two models: One regular one and one with feature dropout to use for the feature importance.

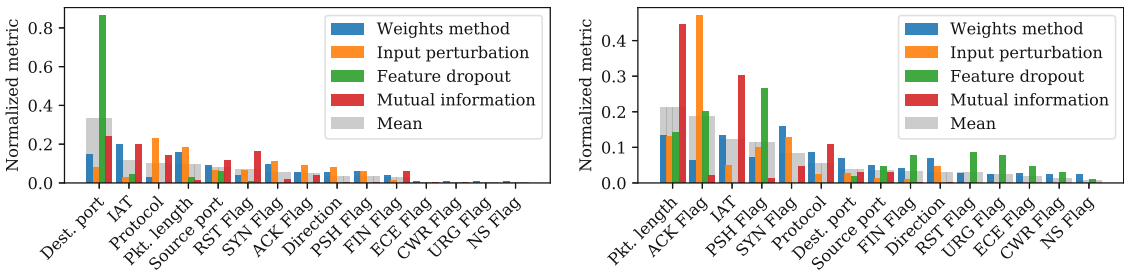


Figure 5.3: Feature importance metrics for the flow prediction for CIC-IDS-2017 (left side) and UNSW-NB15 (right side).

Another method that uses dropout for feature importance is [66], applying a technique called *Variational Dropout* to learn the optimal dropout rate for each feature. It tries to leave out as many features as possible and at the same time keep accuracy high. Thus, important features are going to be left out less often and one can then extract the dropout probabilities for each feature and assess their significance based on them. While this method looks seemingly related to feature dropout, it is significantly more complex and does not aim to show the accuracy drop that occurs when omitting a feature but instead returns a unique feature importance value between 0 and 1.

For feature dropout, it can also happen that several features are left out for a sample and so feature dropout can also be used to analyze the effect of multiple features missing and can thus show possibly correlated features or – more general – features that contain common information, relevant for the classification task: For features i, j we define

$$\text{Score}_{i,j} = \frac{\text{Acc}_{\text{base}} - \text{Acc}_{-i,j}}{(\text{Acc}_{\text{base}} - \text{Acc}_{-i}) + (\text{Acc}_{\text{base}} - \text{Acc}_{-j})}, \tag{5.2}$$

where Acc_{base} denotes the accuracy of the classifier with all features included, with Acc_{-i} the accuracy if feature i is omitted and with $\text{Acc}_{i,j}$ the accuracy if both features i and j are omitted. Assuming a non-negative accuracy drop when omitting a feature, the resulting score is ≥ 0.5. The higher it is, the larger the information that both features share.

For instance, the obtained score between RST and the protocol identifier is 8.5, which is the highest value we observed for all pairs of features. While this may be unintuitive at the first glance, it likely stems from the fact that if the protocol identifier (TCP or UDP) is missing, then RST being 1 at some point indicates that the flow is TCP.

Feature dropout might constitute a building block for methods based on Shapley values [202] like KernelSHAP [158].

Mutual Information Input perturbation and feature dropout mainly address feature importance. For example, assuming that the test dataset is representative for production use, for feature importance it is reasonable to equate the distribution for perturbing a

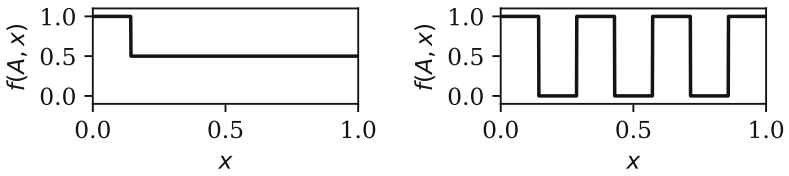


Figure 5.4: Two distributions yielding an identical accuracy drop.

feature with the feature distribution itself. However, when evaluating feature sensitivity, e.g. for analyzing potential for adversarial samples, the attacker is not limited by this distribution and frequently is able to choose arbitrary values in the feature space.

Furthermore, we argue that accuracy drop depicts an importance measure that might be misleading for evaluating feature sensitivity. To see this, let $f(A, x)$ denote the joint probability for classification as attack and a feature value x. Accuracy then is $\int_{\mathbb{R}} f(A, x)dx$ for attacks and $1 - \int_{\mathbb{R}} f(A, x)dx$ for benign traffic. Figure 5.4 shows two different distributions yielding the same accuracy, but clearly the right-side distribution has a larger influence on the prediction, as in the right-side case the prediction deterministically depends on the feature value.

To capture such dependencies, we propose to use mutual information to determine feature sensitivity. Mutual Information between two random variables X, Y is defined as

$$I_{X,Y} = \mathbb{E}\left\{\log\left(\frac{f_{X,Y}(x,y)}{f_X(x)f_Y(y)}\right)\right\}, \tag{5.3}$$

with $f_X(x), f_Y(y)$ and $f_{X,Y}(x, y)$ denoting the distribution of X, Y and their joint distribution, respectively. To obtain feature sensitivity, we compute mutual information between an input variable and the prediction output for one flow at one particular time step, averaging over the result for the test set.

Comparison Figure 5.3 shows the results, which match largely with domain-specific expectations for classifying network flows. In particular, rarely used TCP flags like NS or URG are unimportant for the classifier. On the other hand, destination port and protocol are essential for characterizing flows by hinting at the type of traffic. IAT and packet length are important for estimating the amount of transferred information and several flags hint at certain attacks like Denial-of-Service (DoS) attacks.

The weights method is able to reveal features with a very low importance to a certain degree, but disagrees with the other methods to a large extent. Less anticipated, however, also input perturbation does not exhibit a considerable correlation with feature dropout. Considering its functioning of completely removing individual features, feature dropout is the most reliable method for evaluating importance for removing features. It is remarkable that none of the other methods is able to depict the distinct peak of importance for the destination port visible for feature dropout in Figure 5.3 and Table 5.3.

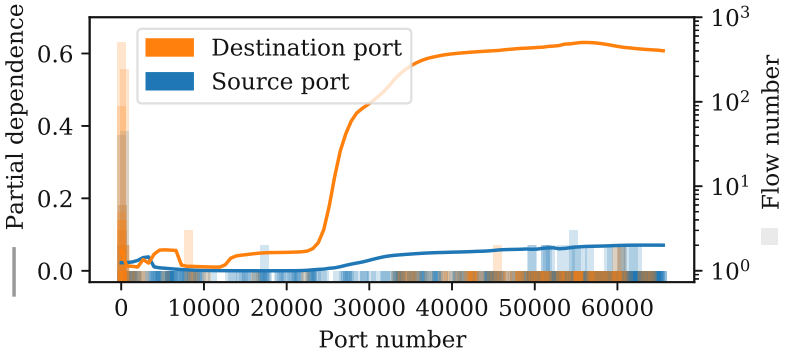


Figure 5.5: PD plot for the source and destination port features.

It is not surprising that mutual information disagrees with feature dropout, since both their aim and their functioning are substantially different. For example, mutual information shows that the protocol field can have a substantial impact on the classification even though an identical accuracy can be achieved when omitting it.

Metrics for UNSW-NB15 differ substantially from CIC-IDS-2017. However, due to the large number of different network attacks and network configurations, it is easily possible that relevant features are very different. We consider the question of model transferability of substantial importance for IDS applications, but out of scope for the present research.

Explainability Plots

Knowing which features to investigate, it is important to analyze which feature values lead to classification as attack. In literature, the use of PDPs has been proposed [92]. To inspect attack types in detail, in this research we use a *conditional* form of the PDP. If $\boldsymbol{X} \in \mathbb{R}^n$ denotes a random vector drawn from feature space, $f(\boldsymbol{X}) \in [0, 1]$ the NN's prediction and c the attack type, we define the conditional PDP for feature i as

$$\mathrm{PDP}_{c,i}(w) = \mathbb{E}_{\boldsymbol{X}|C}\Big(f\left(X_1, \ldots, X_{i-1}, w, X_{i+1}, \ldots X_n\right)|c\Big), \tag{5.4}$$

empirically approximating the conditional distribution by the observed subset that belongs to a certain attack type.

By using a classical PDP we would likely lose information due to the averaging over very different types of traffic. However, for network traffic in particular, investigating each sample individually is not possible. Hence, the conditional PDP provides the ideal compromise for our application.

Due to the use of a 5-tuple flow key, port numbers and the protocol identifier are constant for all packets in one flow. Hence, we can consider the RNN a regular classifier and reuse PDPs, which have been proposed for non-recurrent classification methods, by plotting the RNN's flow prediction outcome over one of these features. The results show that for some attack types the port numbers play an important role. When looking at the PDP

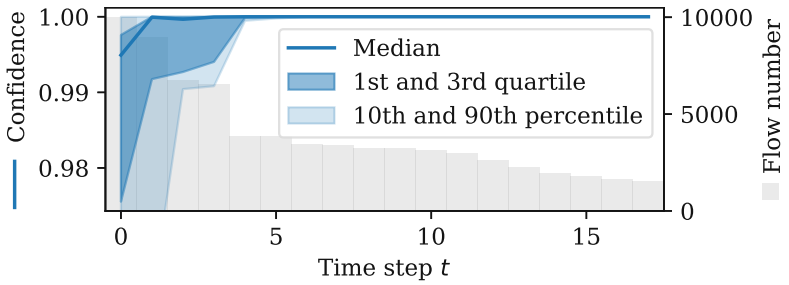


Figure 5.6: Classifier confidence per time step for CIC-IDS-2017. For the majority of attack types, confidence increases in the first few steps and then stays almost constant at 1.

for benign traffic samples in Figure 5.5, it becomes apparent that traffic destined to a high destination port is generally indicative of an attack. We argue that this is because most services that regular users use have low port numbers.

Plots for Sequences

Intuitively, features at the beginning of a flow should be the most important, while the classifier's predictions should not vary significantly anymore, as soon as it has come to a decision.

To evaluate this hypothesis, Figure 5.6 depicts the classifier's prediction confidence for each time step, along with the number of samples having at least this length, which were used for evaluating the figure. While at the first couple of packets the confidence is not very high, towards the end of the flow it reaches values close to 1 and stays there. Hence, not only is the classifier able to make a reasonable classification after just a few packets, the figure also suggests that indeed later packets have a negligible influence on the prediction.

For investigating in more detail how features influence the prediction outcome, we extend PDPs to the sequential setting. Denoting as $X = \{\boldsymbol{X}^1, \ldots, \boldsymbol{X}^m\}$ the series of packet features $\boldsymbol{X}^t \in \mathbb{R}^n$ and $h_t(X)$ the network's hidden state after time step t, we define the sequential PDP as

$$\begin{aligned} &\text{seqPDP}_{c,i}(t, w) = \\ &\mathbb{E}_{X|C}\Big(f\left(h_{t-1}(X), X_1^t, \ldots, X_{i-1}^t, w, X_{i+1}^t, \ldots X_n^t\right)|c\Big). \end{aligned} \tag{5.5}$$

Figure 5.7 shows an example together with the mean values for both unmodified samples and adversarial samples. Also in this figure, we notice that mainly the first few packets are able to influence the prediction outcome and modifying features at a later time step does not change its confidence any longer. In many cases, the adversarial sample generation indeed moves packet features to areas where the network is less likely to be classified as attacks, confirming the effectiveness of PDPs. In other cases, however, we

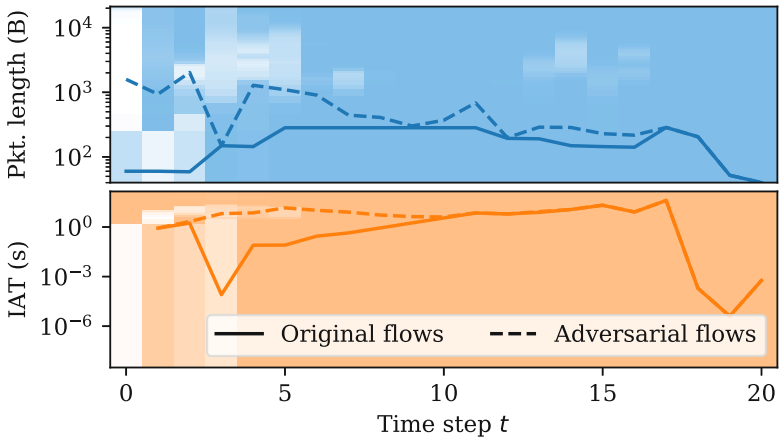


Figure 5.7: Exemplary sequential PD plot and adversarial flows for the DoS Slowloris attack in CIC-IDS-2017. The lines show the feature's mean values. The shaded region shows the change in confidence that occurs when the feature is varied.

did not observe an agreement between PDP and adversarial modifications, hinting at dependencies that cannot be presented by PDPs. Since the IAT is undefined for the first packet, we always set it to 0. Still, interestingly, the plot shows that the classifier considers packets with a higher IAT to be more likely to be attacks than those with a smaller one.

Finally, we investigated whether our classification task involves recognizing complex patterns in the feature space. As example, Figure 5.8 shows that attack types indeed have a characteristic pattern in which they send packets by which they are easily recognizable. Other attack families similarly show characteristic patterns.

5.3.4 Discussion

We have implemented an RNN-based IDS and showed that it is able to detect attacks already at an early point in a flow's lifetime while being able to achieve comparable classification accuracy. The problem of adversarial examples is highly relevant in the

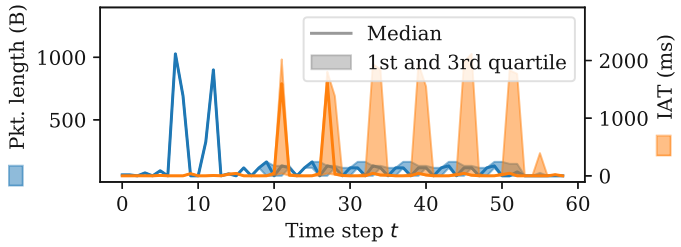


Figure 5.8: IAT and packet length for SSH brute-force attacks in CIC-IDS-2017.

context of IDSs, underlining the question whether the susceptibility to adversarial examples can be analyzed using explainability methods.

Indeed, our sequential PDPs showed that in some cases nearby regions of classification as normal are actively exploited by the adversarial example generation process. Owing to the high-dimensionality of the investigated data, however, in other cases we could not identify a clear relation between the behavior indicated by PDPs and the generated adversarial examples.

As our systematic review of problems related to the interpretation of an RNN-based NN revealed, indeed there are various challenges when it comes to understanding the decision process of such kinds of classifiers. Our conclusion for use in practice, in particular for when security and protection against adversarial ML are of crucial importance, is that other classifiers that are better interpretable are a better choice in many cases.

On the other hand, if RNN-based NNs should be used to profit from their benefits, we recommend analyzing the model to be used using explainability methods before taking it to operation. A systematic way to approach this task is to first use feature importance or sensitivity methods to analyze which features influence the prediction outcome or, on the other hand, which features might even be omitted. As a second step, it is then advisable to perform a more in-depth analysis using visualization techniques as outlined above.

CHAPTER 6

Attacks on High-Security Infrastructures

Notice of adoption from previous publications (Chapter 6)
Parts of the contents of this chapter have been published in the following paper:

> *[118] Alexander Hartl, Joachim Fabini, Christoph Roschger, Peter Eder-Neuhauser, Marco Petrovic, Roman Tobler, and Tanja Zseby. Subverting counter mode encryption for hidden communication in high-security infrastructures. In* The 16th International Conference on Availability, Reliability and Security*, pages 1–11, 2021*

I performed development of the theoretic foundations, development and academic writing by myself. Preparation of figures was performed in collaboration with Joachimi Fabini. All authors contributed in preliminary discussions on attack design and in proofreading of the manuscript.

Chapter 4 showed us that even when using comprehensive encryption, an attack surface remains that might allow analyzing captured network traffic based on observed traffic patterns. However, in RQ4 we went one step further and asked whether the use of encryption might even pose security risks itself. To complete our discussion on RQ4, in this chapter we have a look at a certain attack type where – despite the undisputed security benefits it yields – encryption indeed also introduces additional risks.

Modern communication networks have to be defended against a wide range of attacks. In particular for critical infrastructures, where networks have to meet high demands in security, attacks in many cases are highly sophisticated. While attacks that incur unusual network communication are detectable using methods highlighted in the previous chapters, it is illusive to believe that any possible network attack can be detected. To

defend against sophisticated attacks, it is thus crucial to implement security measures also on the organizational layer of a company, e.g., by educating employees on security-related topics. To motivate these claims, in this chapter we construct an attack whose detection is computationally infeasible despite being able to observe the relevant traffic. This attack we outline in this chapter implements network steganography by using a subliminal channel.

The concept of subliminal channels was introduced as early as 1978 and first described publicly by Simmons in 1984 [210, 207]. Simmons envisioned two prisoners who are allowed to communicate with each other in the form of messages. Since the prison warden wants to prevent the prisoners from coordinating an escape plan, he only relays messages that are unencrypted so that he can read them himself. The prisoners, on the other hand, fear that the warden will forge each other's messages, so they insist on using signatures to authenticate the communications. Using this scenario, Simmons showed that information can be embedded in a signature that does not interfere with successful signature verification.

So far, the existence of subliminal channels has been shown for many traditional signature schemes, such as DSA ([209], [78]), ECDSA ([55, 75, 137]), RSA ([244, 53, 189]), or Rabin signatures ([210, 184]). Finding a method to use a signature scheme provably without the possibility of a subliminal channel usually turns out to be extremely difficult and requires the use of a central authority that can monitor the communication, which is hardly feasible for many practical scenarios [115].

Unlike the majority of work on subliminal channels, we here investigate a symmetric cipher for the possibility of covert communication. In more detail, we exploit the Message Authentication Code (MAC) of AES-GCM-encrypted data to carry information. While the possibility of exploitation of a MAC as carrier of covert information is not particularly interesting when using the cipher in a straightforward way, we here investigate it in the context of a network architecture that is explicitly designed to prevent cryptographic attacks by offloading cryptographic operations to a separate network device with paramount security properties. Although intuition suggests that the alleged paramount security of such an architecture prohibits the exploitation of the MAC, we show that shortcomings of GCM indeed allow use as a subliminal channel.

6.1 Related Work and State of the Art

GCM was proposed by McGrew and Viega [162] to overcome performance restrictions in processing rates of used encryption modes while providing provable security. The authors included a security proof in their original publication [162], but Iwata et al. [135] later uncovered a flaw in the security proof. Even though further security issues were discovered for GCM [193, 112, 138, 84, 104], GCM is generally being considered secure when used correctly. In particular, when showing that the original security proof is faulty, Iwata et al. additionally provided a new proof, which, however, has larger bounds. Also multi user security of GCM has been shown [46, 123].

In the present context, Joux's attack [138], termed the "forbidden attack", is noteworthy. The attack's name stems from the fact that it relies on Initialization Vectors (IVs) being reused, which is a scenario that is explicitly excluded in security proofs. In this case, security of GCM breaks down completely and universal forgery becomes possible. Later, Böck et al. [61] performed a scan of the Internet, showing that, at the time of their studies, a notable amount of TLS-protected webservers suffered from implementation issues, which led to IV reuse.

In this thesis, we are interested in exploiting GCM's properties for hidden communication. The exploitation of cryptography for hidden communication was originally introduced by Simmons [207, 208], who envisioned prisoners who communicate clandestinely using authentication data, and introduced the term "subliminal channel" for hiding information exploiting cryptographic algorithms. To date, subliminal channels have been found in various digital signature schemes [209, 56, 54, 114]. The existence of subliminal channels has also been shown in the field of post-quantum cryptography, where principles are used that are often fundamentally different from more traditional cryptography [115, 93, 149]. The field was extended by Young and Yung [238, 239], terming the field "kleptography" and introducing Secretly Embedded Trapdoor with Universal Protection (SETUP) attacks.

More recently, approaches in the spirit of subliminal channels and SETUP attacks have been investigated under the name Algorithm Substitution Attack (ASA) [47] and it has been investigated how randomness used for symmetric encryption can be used for leaking information [47, 48, 36]. For example, Armour and Poettering [36] showed how Authenticated Encryption with Associated Data (AEAD) encryption can be used to leak information by raising spurious authentication failures, encoding information in the resulting message retransmission pattern. Schneier et al. [199] survey various methods for weakening cryptographic algorithms.

6.2 Covert Communication using AES-GCM

In high-security infrastructures, use of cryptographic methods is ubiquitous to ensure confidentiality, integrity and authenticity of transmitted data. The most frequent requirement is to provide both, message confidentiality and authenticity, which has led to the development of AEAD encryption modes. An AEAD algorithm performs encryption of the data and additionally computes an authentication tag, thus providing both, confidentiality and authenticity.

As of today, the most popular method to perform authenticated encryption is to use GCM, utilizing AES as underlying block cipher.

Additional security improvements can result from restricting involved parties' direct access to secret values. Tasks like encryption, decryption and authentication can be offloaded to a dedicated device like, e.g., a separate network server or a dedicated hardware module. In this thesis, we denote such devices as Cryptographic Key Management

Devices (CKMDs). Secret keys, which are considered among the most valuable pieces of information to be kept confidential, can thus be stored in a single location that is equipped with paramount security features. Architectures deploying CKMDs are often used within critical infrastructures, aiming to meet highest demands in security and privacy.

When designing a high-security infrastructure it thus seems natural to combine both, AEAD encryption and the use of CKMDs. However, as we show in this chapter, the straightforward use of counter mode encryption such as GCM, CCM, CWC, EAX or the plain Counter Mode (CTR) in such a scenario opens doors for attack options that allow the circumvention of the CKMD in encryption and decryption tasks, cancelling a large part of benefits the CKMD provides.

For example, since a message's authentication tag does not carry actual information, it might be exploited by malware to carry hidden information, which can pose a major threat in a critical infrastructure, as it evades detection possibilities by network monitoring software and IDSs. When using a CKMD, it seems easy to prevent this communication possibility by denying decryption on the CKMD for messages with invalid authentication information. We show that, when using counter mode encryption like GCM, attack possibilities can emerge that allow circumvention of the CKMD's decryption interface in both decryption and authenticity verification, thus allowing the authentication tag to be used as a very attractive subliminal channel. Due to its immense practical relevance, we focus on GCM in this work.

Hence, in this chapter, we review attack possibilities that emerge from using GCM in conjunction with a CKMD. We analyze methods for exploiting the authentication tag for the establishment of a subliminal channel, which yields particularly attractive properties for an attacker with respect to hidden communication. To high-security infrastructures, this subliminal channel therefore poses a major risk. We additionally discuss approaches to remedying the risk originating from the described scenarios, concluding that the use of a different operational mode like GCM-SIV or of random, CKMD-generated Initialization Vectors (IVs) are the best approaches to avoiding attack possibilities.

Our Contributions

With our findings in this chapter, we aim to raise awareness for a false sense of security arising when combining counter mode encryption like GCM with architectural design deploying CKMDs. This finding is of particular significance, because both, GCM and the strategy of offloading cryptographic tasks to CKMDs are known to provide high security, and hence, are likely to be used in critical infrastructures where the existence of hidden communication facilities can be detrimental for privacy, security or even safety.

In this case, the CKMD seems to provide means for restricting the possibilities for cryptographic operations that can be performed. As we will highlight presently, however, such restrictions can easily be bypassed in many cases when using counter mode encryption.

We discuss in detail a scenario where the CKMD allows both encryption and decryption of arbitrary messages and investigate whether the authentication tag of GCM can be abused for hidden communication while leaving the actual communication operational. Intuitively, a CKMD would be expected to insist on a correct authentication tag for performing decryption, leaving no space for embedding hidden information. We show that this intuition is misleading and that it is indeed possible to perform decryption despite a manipulated authentication tag. To demonstrate that our used scenarios are relevant in practice, we describe an exemplary scenario in the field of the Industrial Internet of Things (IIoT) and show that the subliminal channel can pose a major risk, while most known covert and subliminal channels can easily be prevented.

GCM is used in various protocols, e.g., in TLS [196, 186], IPsec [222], MACsec [14] and SSH [134]. Due to its immense practical relevance, we thus focus on GCM in this thesis. However, while our technique for hidden communication seems to be very specific for GCM, in fact also other counter encryption modes like CCM [229], CWC [145] and EAX [45] follow an architectural design that allows applying techniques as described here for circumventing CKMDs to hide information in an authentication tag. We also provide several reasons for why, assuming a CKMD is deployed, it is likely that it is deployed in a way that allows attacks as described by us. Furthermore, our concept is generic in the sense that the sender of hidden information does not have to be co-located with the sender of overt communication, but instead can be located on any node on the transmission path that is able to modify the encrypted message. This property allows various scenarios for exploitation by an attacker.

To provide guidelines for protocol engineering, we finally discuss several approaches to mitigating attack possibilities. Beside the use of a different mode of operation like GCM-SIV [105], the most potent approach to avoid attack possibilities described in this thesis is to generate the IVs on the CKMD, even if the IVs have to be chosen at random in this case due to limited capabilities of a CKMD. This is particularly interesting, as it has been shown previously that the use of random IVs instead of counting IVs *increases* susceptibility for hidden communication [47].

6.2.1 Offloading Cryptographic Tasks

In many architectures, cryptographic operations are offloaded to a specific device, in this thesis referred to as CKMD. We introduce the term CKMD as an abstract term, which in practice occurs in different instantiations. In particular, a CKMD might be

- a dedicated network server, which performs, e.g., encryption, decryption or signature creation tasks for other network participants.
- a Trusted Platform Module (TPM), i.e. a dedicated device soldered on computers' mainboards to provide a secure key storage for cryptographic tasks.
- a Smart Card, functioning as portable security token.

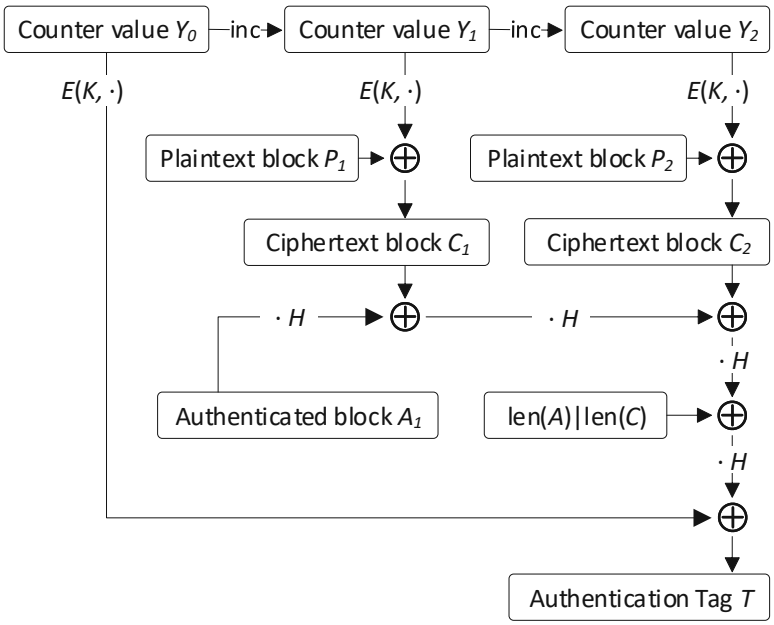


Figure 6.1: The GCM encryption process for two blocks of encrypted data and one block of additional authenticated data.

The main reason for deploying a CKMD is to provide means to store and encapsulate cryptographic key material in a highly secure manner and restrict access to cryptographic functions using it to a well-defined, security and privacy-preserving interface. In particular, often special hardware features are used to provide a secure storage area shielding keys from unauthorized access, possibly even despite physical access. A CKMD might additionally be equipped with special hardware for performing cryptographic tasks, like, e.g., a good source of randomness or hardware support for cryptographic algorithms.

Furthermore, a CKMD might be used to restrict the usage of cryptographic keys or monitor how they are used as well as the frequency of usage. For example, a CKMD might permit data to be encrypted, but prohibit the decryption of data, even though a symmetric key is used. However, as we show in the following, possibilities for restricting usage might be easily bypassed when using an operational mode like GCM.

6.2.2 GCM Encryption

Frequently it is necessary to both encrypt and authenticate transmitted messages. Encryption modes that provide both functionalities are known as AEAD. A very popular AEAD encryption mode is GCM. We summarize GCM encryption in this section.

Encryption and Decryption

When processing 128-bit plaintext blocks P_i with $1 \leq i \leq n$ with a user-provided IV, GCM produces ciphertext blocks C_i and an authentication tag T. The encryption operation of GCM is defined by the following set of equations [162]:

$$
\begin{aligned}
H &= E(K, 0^{128}) \\
Y_0 &= \begin{cases} \text{IV}|0^{31}|1 & \text{if len(IV)} = 96 \\ \text{GHASH}(H, \{\}, \text{IV}) & \text{otherwise} \end{cases} \\
Y_i &= \text{inc}(Y_{i-1}) \quad \forall\, i = 1, \ldots, n \\
C_i &= P_i \oplus E(K, Y_i) \quad \forall\, i = 1, \ldots, n-1 \\
C_n^* &= P_n^* \oplus (E(K, Y_n)[: \text{len}(P_n^*)]) \\
T &= (\text{GHASH}(H, A, C) \oplus E(K, Y_0))[: L_T]
\end{aligned}
$$

Here, $E(\cdot)$ denotes encryption using the underlying block cipher (e.g. AES [7]), K denotes the symmetric key used for encryption, $\oplus$ denotes the Exclusive Or (XOR) operation or, equivalently, addition in $\text{GF}(2^{128})$ and L_T denotes the desired length of the authentication tag T in bits. Furthermore, we denote by $x[: N]$ the N most significant bits of x and by $x[N :]$ the N least significant bits of x. $\text{len}(\cdot)$ returns its argument's length in bits and $\text{inc}(\cdot)$ returns its argument incremented by one in big-endian representation.

The function GHASH() is obtained from X_i defined according to the equations [162]

$$
X_i = \begin{cases}
0^{128} & \forall\, i = 0 \\
H \cdot (X_{i-1} \oplus A_i) & \forall\, i = 1, \ldots, m-1 \\
H \cdot (X_{i-1} \oplus (A_m^* | 0^{128 - \text{len}(A_m^*)})) & \forall\, i = m \\
H \cdot (X_{i-1} \oplus C_{i-m}) & \forall\, i = m+1, \ldots, m+n-1 \\
H \cdot (X_{i-1} \oplus (C_n^* | 0^{128 - \text{len}(C_n^*)})) & \forall\, i = m+n \\
H \cdot (X_{i-1} \oplus (\text{len}(A) | \text{len}(C))) & \forall\, i = m+n+1,
\end{cases}
$$

where $\text{GHASH}(H, A, C) = X_{m+n+1}$. Here, $\cdot$ denotes multiplication in $\text{GF}(2^{128})$ and A_i denote 128-bit blocks of additional authenticated data with $1 \leq i \leq m$. The encryption operation is illustrated in Figure 6.1. The transmitted message then consists of the ciphertext blocks C_i and the authentication tag T.

Hence, similar to stream ciphers, encryption is done by performing an XOR operation of the plaintext with the block cipher's output. Decryption can be performed by again performing an XOR operation with the ciphertext, eventually allowing computation of the authentication tag similar to the encryption operation.

AES-GCM is a very popular cipher. For example, it is used in TLS [196, 186], IPsec [222], MACsec [14] and SSH [134]. In TLS version 1.3 AES-GCM is the only cipher that has to be implemented by standard-compliant implementations [186].

Reasons for the widespread use of AES-GCM boil down to the reasons that motivated the development of GCM in the first place. In particular, due to the way GCM is engineered, encryption and decryption are parallelizable and can be efficiently implemented in hardware and software [162, 163], thus allowing vast amounts of data to be processed and deployed in high rates. The key stream used for encryption and decryption can be precomputed in advance, which constitutes a convenient feature for many applications. Furthermore, GCM is provably secure [135] and was designed to be free of patents [162].

IV Selection

As will become apparent later, it is of utmost importance to avoid encrypting distinct messages with the same IV. In general, IVs can be chosen randomly or deterministically by incrementing the IV for each processed message by one. For example, the NIST recommendation [77] allows the following methods for creating IVs:

For deterministic construction, [77] divides the IV field into two subfields, the fixed field and the invocation field. The fixed field identifies the context for the encryption, so that, e.g., for each device a unique value is chosen. For a specific value of the fixed field, the invocation field is then simply incremented for each processed message or generated using a linear feedback shift register.

For Random Bit Generator (RBG)-based construction, a random IV is generated for each processed message. If the IV is chosen at random, the uniqueness of occurrences of IV values cannot be guaranteed. Hence, [77] defines an upper limit of 2^{-32} for the probability of ever assigning the same IV to distinct messages, thus limiting the number of processed messages with the same key. For example, for an IV length of 96 bits, up to 2^{32} messages can be processed with the same key without violating the requirement.

The benefit of RBG-based construction is to avoid having to keep a state for the encryption operation and the risks of flawed state keeping. However, for RBG-based construction only a probabilistic statement can be made for the reoccurrence of IVs. It is also noteworthy that for the same number of processed messages, the IV needs to be considerably longer than for deterministic construction. Furthermore, if both sender and receiver can be constructed stateful, IV values do not even need to be transmitted for deterministic construction, as the receiver can construct them on his own. Furthermore, the deterministic construction has been argued to reduce facilities for hidden communication [47].

For these reasons, deterministic construction becomes increasingly popular. For example, in IPsec [222] and TLS version 1.2 [196] IVs are transmitted with each message, leaving selection of suitable values to the implementation, but suggesting simply using a counter. In TLS version 1.3 [186] and SSH [134], IVs are no longer transmitted, but implicitly computed based on a counter.

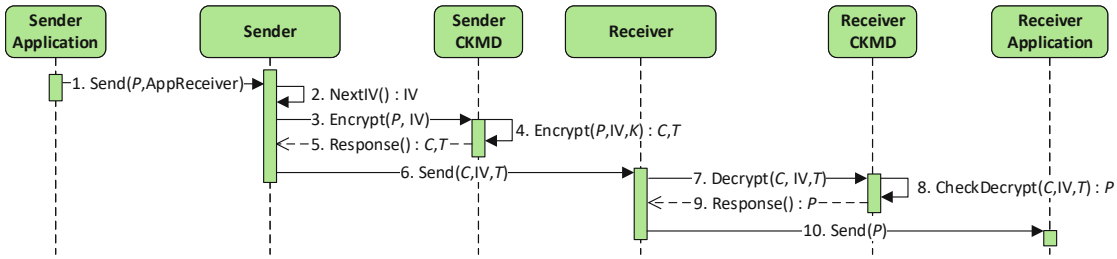


Figure 6.2: Normal device communication when using CKMDs.

6.2.3 CKMD Architectures and Counter Mode Encryption

As described in the previous Section 6.2.2, IV selection frequently takes place deterministically by incrementing a counter for each processed message. When deploying a CKMD, the question arises whether to implement this counting procedure on the CKMD itself, or rather on the requesting device, transmitting it to the CKMD within the encryption request.

We argue that in most practical scenarios the counter indeed is implemented on the requesting device for reasons of robustness, cost, simplicity and hardware design. On the one hand, CKMDs in many cases are low-cost devices, where state keeping is difficult. On the other hand, even if state keeping is possible, robustness of the communication might be at stake or would at least require additional complexity when computing deterministic IVs on the CKMD. For example, if the connection between requestor and CKMD is unreliable, the delivery of an encryption response might fail. In this case, if the state on the CKMD has been updated anyhow, a wrong IV would be the consequence when rerequesting encryption, leading to failing decryption if the IV is computed implicitly on the receiver side.

We thus argue that in practice a reasonable scenario is that the IV is chosen by the requestor and submitted to the CKMD for each message, leaving the requestor full control over which IV is used. However, if the requestor is in full control about the used IVs, it can perform several operations described in this section, which are usually implicitly considered impossible.

Decrypting Messages

A CKMD might allow the encryption of messages, but prohibit the decryption of messages. If counter mode encryption modes are used, such constraints are elusive. Since counter mode encryption modes function like stream ciphers with respect to encryption and decryption, both encryption and decryption proceed by performing an XOR with a key stream obtained from the underlying block cipher. Hence, decryption of an encrypted message C_i can be performed by instead requesting encryption with the same IV, interpreting the returned ciphertext as decrypted plaintext.

However, the CKMD might monitor the messages to be encrypted and enforce a certain message syntax and semantics. This check can be circumvented as well by requesting the encryption of an arbitrary, syntactically valid, message R_i with the same IV, resulting in the ciphertext $C_{R,i}$. Decryption of the original message can then be performed by computing the XOR $P_i = C_i \oplus R_i \oplus C_{R,i}$.

This attack strategy might also be used for performing decryption independent of the authenticity of the message. If the CKMD determines that an encrypted message's authentication tag is wrong, it usually not only reports the failing authenticity check, but also denies decryption of the message. In Section 6.2.4, we will make use of this attack to decrypt messages while exploiting the authentication tag to carry hidden information.

The attack scenario described in this section is not limited to GCM, but rather constitutes a general weakness of counter mode encryption. In particular, it similarly applies to plain CTR [3] encryption, but also to the authenticated CCM [229], CWC [145] and EAX [45] encryption modes. It does not apply to the recent GCM-SIV [105] and AES-GCM-SIV [106, 103] modes, however, as the IV used for counter mode encryption is derived from the message itself for GCM-SIV and AES-GCM-SIV. Due to the practical relevance of GCM, we focus on GCM in this thesis.

Encrypting and Authenticating Messages

Similar to the previous Section 6.2.3, an attacker is able to obtain an encrypted message C_i from plaintext P_i by requesting encryption of a different message of equal length instead. However, for an attacker this approach is only useful if he also is able to generate a valid authentication tag T.

From one observed ciphertext/plaintext pair an adversary can form the polynomial $c_1 H^{m+n+1} + \ldots + c_{m+n} H^2 + (\text{len}(A)|\text{len}(C))H + E(K, Y_0) = T$, where the coefficients c_i can be derived from the non-secret A_i and C_i. To authenticate arbitrary messages, the attacker needs to know H and $E(K, Y_0)$, which cannot be deduced from a single polynomial equation. However, as soon as a second message for the same IV is observed, the equation system becomes solvable, obtaining $E(K, Y_0)$ and a small number of candidates of H. Hence, by requesting the encryption of two messages with equal IV, it is possible to circumvent the CKMD in the creation of authenticated messages. This attack is known as the "forbidden attack" [61], as it relies on IVs being reused, which is a scenario that is explicitly excluded in security proofs for GCM.

Note, however, that the sole possibility for decryption does not enable the attacker to authenticate arbitrary messages, since he cannot obtain distinct plaintext pairs with the same IV in this case. He can, however, if encryption is allowed, easily circumvent restrictions with regards to the content or the amount of encrypted messages.

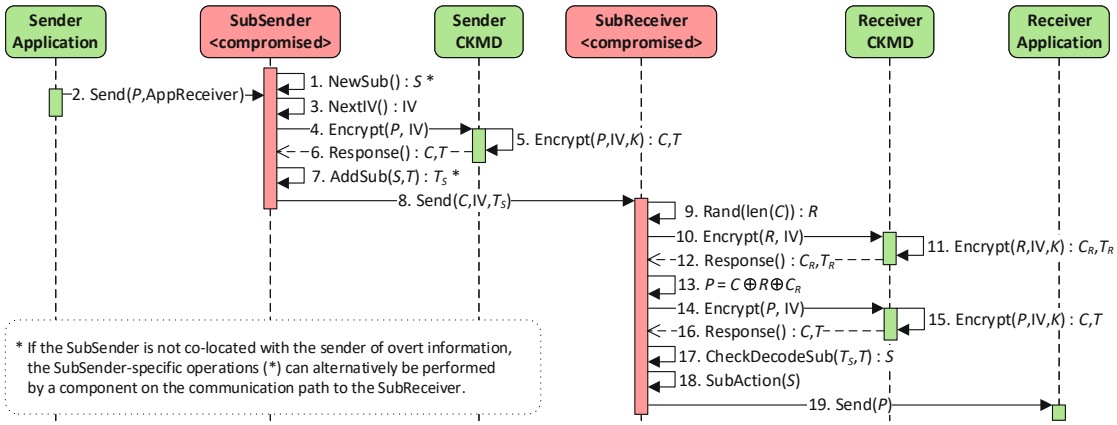


Figure 6.4: Subverting CKMDs in building a subliminal channel.

6.2.4 Exploiting GCM for Subliminal Communication

We now describe how the use of GCM can lead to a subliminal channel for architectures where CKMDs are used. We target particularly high-security infrastructures, where deployed network protocols have been designed to minimize the possibility for covert channels and possibly even the use of digital signatures, which are prone for subliminal communication [209, 56, 54, 114], might have been avoided. At the same time, in security-critical infrastructures, the need for ensuring message integrity and authenticity is inevitable, leading to the use of AEAD encryption like GCM.

Using T for Subliminal Communication

Figure 6.2 shows device communication for normal, i.e. benign, operation. Here, both sending and receiving parties are equipped with CKMDs, which perform the actual encryption and decryption tasks. When the sending application transmits a message P, the Sender creates a new IV, passes both P and the IV to the CKMD for encryption in step 3 and sends the returned ciphertext C and authentication tag T to the Receiver in step 6. The Receiver passes C, T and the IV on to the CKMD in for decryption step 7, which verifies message authenticity and performs decryption. In case of successful verification of the authentication tag T, the CKMD returns the message P to the Receiver in step 9. Finally, the receiving application obtains the message P. If verification of T fails, the CKMD denies decryption and returns a decryption error.

A Trivial Covert Channel We now describe how a subliminal channel can be established, exploiting the authentication tag for carrying hidden information. To this end, without going into detail about how compromisation might take place, we assume that the Sender and Receiver have been compromised and require means to communicate clandestinely thereafter. As we will demonstrate in Section 6.2.5, it is easily possible that Sender and Receiver cannot rely on covert channels of lower network protocols and,

hence, can only use the message itself or its authentication tag for hidden communication. Sender and Receiver deploy CKMDs, which we do not assume compromised. Hence, Sender and Receiver do not have access to secret keys for encryption.

With this setup, we will encode hidden information into the authentication tag of encrypted messages in what follows. Hence, embedment of hidden information would cause the legitimate verification procedure to fail. While this seems unsuitable for subliminal or covert channels at first sight, since the encryption key is needed for verifying message authenticity, only Sender and Receiver can verify message authenticity. In our case, both Sender and Receiver are compromised, as they perform the hidden communication, and therefore do not report the existence of the subliminal channel.

Of course, someone in possession of the respective secret key might reveal existence of the subliminal channel. However, due to the secrecy of the appropriate key, this scenario is highly unlikely. Furthermore, assuming that random oracles are implemented using cryptographic hash functions, also traditional subliminal channels can be unveiled when holding the secret key.

To ease comprehension, we begin by presenting how a covert channel can be established when neither of the communicating parties deploy CKMDs. Figure 6.3 shows a scenario where hidden information is transmitted from the SubSender to the SubReceiver. In this scenario, it is possible to exploit the authentication tag for hidden information without exploiting the mechanisms described earlier in this chapter.

Steps 1-4 in Figure 6.3 match normal operation, where SubSender obtains a message from the application and performs authenticated encryption, obtaining C and T. However, to clandestinely transmit information, SubSender encodes hidden information in step 5. The operation AddSub() accepts the authentication tag T and computes the altered authentication tag T_S, so that $T_S[L_T - L_S :] = T[L_T - L_S :] \oplus S$ and $T_S[: L_T - L_S] = T[: L_T - L_S]$, where L_S denotes the length of the hidden information S. Hence, the hidden information is encoded in the L_S least significant bits of T using an XOR operation.

The message C, T_S is then sent to the SubReceiver in step 6, which can perform decryption and computation of the expected authentication tag T in steps 7,8 as usual. However, instead of raising an authentication failure when the received tag does not match T, authenticity of the message is verified by only comparing the most significant $L_T - L_S$ bit of T. Furthermore, hidden information $S = T_S[L_T - L_S :] \oplus T[L_T - L_S :]$ can be recovered in step 9 in terms of an XOR with the L_S least significant bits of the authentication tag computed during decryption.

Circumventing CKMD Verification We now assume that both SubSender and SubReceiver deploy non-compromised CKMDs, aiming to achieve superior security. Receiving a message with a faulty authentication tag, a CKMD is likely to deny message decryption and possibly even raise an alarm. Exploitation of the authentication tag as subliminal

channel therefore does not seem feasible anymore without sacrificing decryptability of the message.

However, it is one of the specific properties of GCM encryption, which allows exploitation as subliminal channel also in this case. Figure 6.4 shows the steps that SubSender and SubReceiver undertake in submitting the message. Hence, in steps 4-6 similar to normal operation, the SubSender submits the message to its CKMD for encryption. After receiving back the ciphertext blocks C_i and authentication tag T, however, the SubSender embeds hidden information S in step 7. Similar to the previous scenario, AddSub() encodes S into the least significant bits of T in terms of an XOR operation, obtaining T_S.

At the SubReceiver, processing of the message diverges from the previous scenario, as the encrypted message cannot simply be passed to the CKMD due to the manipulated authentication tag. As described in Section 6.2.3, the SubReceiver can perform decryption by issuing an encryption request. Hence, in step 9 the SubReceiver generates an arbitrary message R that has at least the same length as C and passes it to the CKMD for encryption in steps 10-12, using the same IV as in the received message. It receives back the ciphertext C_R and authentication tag T_R. Decryption of the ciphertext blocks C_i can now be performed in step 13 as $P_i = C_i \oplus R_i \oplus C_{R,i}$.

For both verifying message authenticity and decoding the hidden information, however, the original authentication tag T is needed. For this reason, the SubReceiver now passes the obtained plaintext blocks P_i to the CKMD for encryption in steps 14-16, obtaining the original authentication tag T. Similar to the previous scenario, the SubReceiver can now recover $S = T_S[L_T - L_S :] \oplus T[L_T - L_S :]$ and verify message authenticity by verifying the $L_T - L_S$ most significant bits of T_S in step 17. If verification passes, the SubReceiver processes S in step 18 and, similar to normal operation, passes P to the receiving application in step 19.

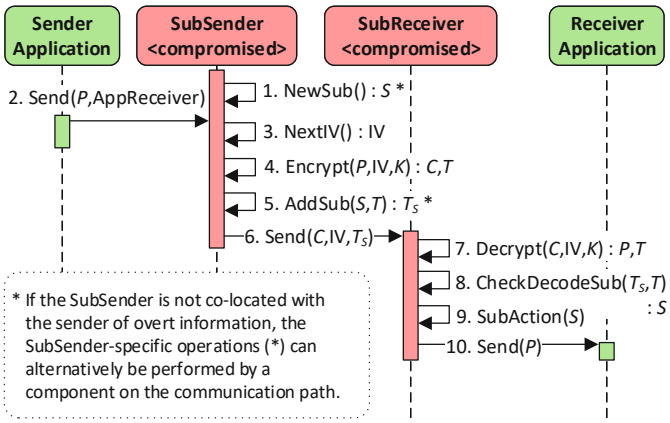


Figure 6.3: Exploiting the authentication tag for hidden communication.

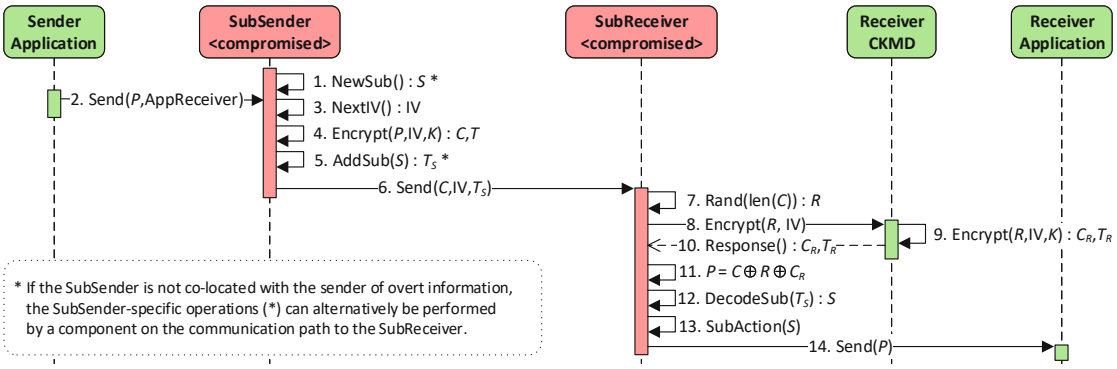


Figure 6.5: A simplified subliminal channel.

Proof of Concept Implementation To verify functionality in practice, we used the Envelope (EVP) encryption interface of OpenSSL version 1.1.1k and implemented the steps depicted in Figure 6.4. We used `EVP_aes_256_gcm` as cipher. The 256-bit encryption key and 96-bit IV were chosen at random. Furthermore, we used a random authentication tag with length between 8 bytes and 16 bytes, a random message with length between 1 and 100 bytes and a random hidden message with length between 1 byte and $\text{len}(T)/8 - 1$ bytes. In all 10^9 runs we performed, verification of authenticity, decryption of the overt message and recovery of the hidden message proceeded successfully.

Properties of the Subliminal Channel

In high-security environments, our presented subliminal channel has properties that might be particularly attractive to an adversary.

Deployment The subliminal channel is agnostic to the used protocol. Hence, an adversary does not have to learn semantics of the used protocol, but only needs to compromise cryptographic algorithms. Since in normal operation preservation of the authentication tag is necessary to ensure successful authenticated decryption, an adversary can be sure that the subliminal information is retained from sender to receiver, independent of how many nodes are in between them. This is in contrast to covert channels in network headers, which might be destroyed by, e.g., devices performing Network Address Translation (NAT).

Concealment Properties As discussed above, discovery of the subliminal channel is possible only if the encryption key is known, which usually is difficult to achieve in practice. Furthermore, even if the existence of the subliminal channel is known, decoding of the information is only possible when holding the encryption key.

Of course, the requests that are sent to the CKMD differ substantially from usual operations. Hence, statistical monitoring on the CKMD, or of the communication with

the CKMD, might show abnormalities. However, considering the characteristics of the architecture, it is highly unlikely that the subliminal channel would raise suspicion. In particular, in most cases the CKMD lacks resources for performing statistical monitoring and performing anomaly detection on the observed data.

Also considering its communication, even if communication can be monitored, e.g., in a network environment, leaving this communication unencrypted would be detrimental for security. Hence, in a practical scenario this communication must be encrypted, e.g., using a TLS-protected transport, making communication monitoring highly challenging.

Bandwidth for Hidden Communication For many practical security protocols, small chunks of information are encrypted and authenticated like, e.g., network packets. Hence, authentication tags are transmitted with a substantial frequency, offering a large bandwidth for subliminal communication. In particular, compared to subliminal channels known from digital signature schemes, the resulting number of requests to the CKMD and the available bandwidth for hidden communication in most cases are considerably higher.

Location of Compromised Devices In Figure 6.4, we depicted a scenario where compromised parties coincide with the sender and receiver of benign communication. Indeed, the receiver of the hidden information can only coincide with the receiver of the overt information, as depicted in Figure 6.4, or, trivially, the receiver's CKMD. The secret key needs to be known to recover the hidden information and, furthermore, authenticity verification would fail if the receiver cannot be compromised.

On the other hand, for encoding the hidden information no secret information is needed. Hence, the sender of hidden information does not necessarily need to coincide with the sender of the overt information, but instead might be located on any intermediate node that is able to manipulate the transmitted message. We also want to note that, when being able to compromise both sender and receiver, it might in certain cases be possible to entirely circumvent CKMDs on both sender and receiver side and deploy an attacker-provided encryption, hence allowing to create other facilities for hidden communication. However, compared to a simple manipulation of the authentication tag, such approach clearly would require a more extensive compromisation of the involved devices, additional measures to avoid detection, and might overload the computational resources of low-power devices like, e.g., sensors, as encryption can no longer be offloaded to the CKMD. Furthermore, in comparison to such scenarios, dropping the requirement for compromising the sender allows the described subliminal channel to be used in various further scenarios. This property of the subliminal channel thus raises its significance substantially, as attackers can choose arbitrary locations on the communication path to unidirectionally communicate hidden information into a high-security infrastructure.

A Simplified Subliminal Channel

In Figure 6.5, we show a variant of the subliminal channel described in Section 6.2.4, which has slightly different characteristics. Steps 1-4 proceed as before. However, when skipping the second encryption request in Figure 6.4, the original authentication tag is no longer available for recovering S or ensuring message authenticity. Therefore, in contrast to Section 6.2.4, in Figure 6.5, the operation AddSub() directly encodes S into the least significant bits of T_S instead of performing an XOR, i.e. $T_S[L_T - L_S :] = S$. In step 12, this information can then trivially be recovered. However, it is no longer possible for the SubReceiver to verify message authenticity.

This variant of the subliminal channel is slightly simpler, as it avoids a second encryption request to the Receiver's CKMD. It might also be of relevance for an attacker if the communication between SubReceiver and the SubReceiver's CKMD could be monitored. In this case, it is unlikely that the observer is able to tell apart encryption requests from decryption requests, since the communication to the CKMD itself is very likely to be encrypted. However, since the processing of a message generates two requests to the CKMD in Figure 6.4 instead of just one, observing the request pattern might already unveil the existence of the subliminal channel.

For the attacker this simplified variant has a few downsides. Beside the fact that T is no longer available to ensure message authenticity, the hidden information S is no longer protected by the encryption key. Hence, when directly encoding transmitted information into S, the information would be openly readable by anyone who is able to intercept the message from the SubSender to the SubReceiver. Furthermore, the distribution of T would deviate from a uniform distribution, disclosing the existence of a subliminal channel on closer analysis. The attacker can counter these drawbacks by performing suitable encryption of S using an attacker-provided key.

Similar to the variant described in Section 6.2.4, we created a proof of concept to verify functionality of this simplified subliminal channel.

6.2.5 An Exemplary Infrastructure

To illustrate the relevance of the facilities for hidden communication described in this chapter, we now discuss an exemplary infrastructure, where the subliminal channel might pose a severe threat to security. Figure 6.6 illustrates a practical implementation of the previous generic architecture. The sensor network at the left comprises many IIoT devices like, e.g., charging stations, smart meters, or smart manufacturing devices. These IIoT sensors must authenticate and encrypt the generated data at application layer before sending it to the sensor network, presumably relying on AES-GCM due to its popularity and attractive properties.

Most IIoT devices have limited resources in terms of computational power and memory because of cost considerations. Sensor devices may benefit from using a CKMD in form of a tamper-proof on-board module over storing the cryptographic key in clear for two

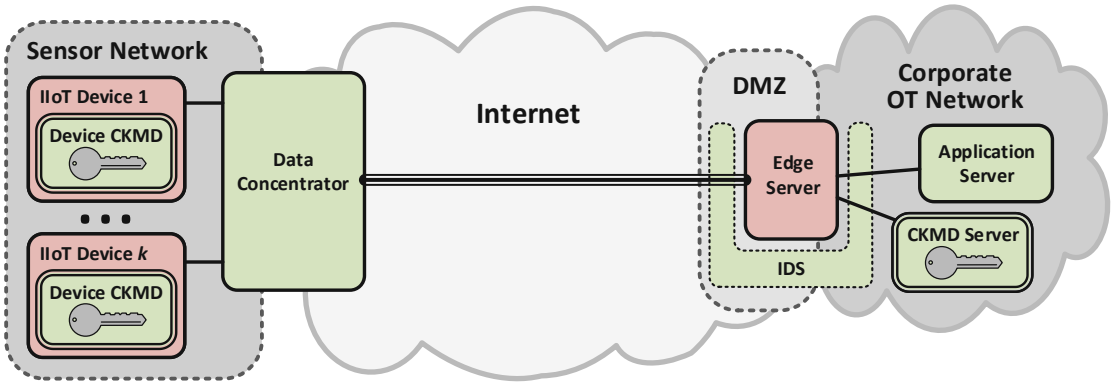


Figure 6.6: Exploiting the subliminal channel in an IIoT infrastructure.

main reasons: Performance and security. From a performance perspective, the offloading of computationally expensive cryptographic operations to dedicated hardware on the CKMD can reduce overall system sizing and minimize costs by using specialized hardware on the CKMD. From a security perspective, the cryptographic key material must be safely protected against eavesdropping, considering that the sensor devices may be subject to continued physical access by customers or, even worse, unauthorized third parties.

The Data Concentrator in Figure 6.6 aggregates the data of several IIoT devices on the sensor network. It maintains a security association (for instance TLS or IPsec) to tunnel the data via the public Internet to an Edge Server connected to the corporate network's Demilitarized Zone (DMZ).

The Edge Server's role is to protect the trusted corporate Operating Technology (OT) network against outer access while serving as a security gateway for data that was generated by external sensors. On this behalf, the Edge Server terminates the Data Concentrator's security tunnel and demultiplexes the aggregated data. Subsequently, it verifies and decrypts the data originated by the IIoT devices using the CKMD Server, i.e. one of possibly several dedicated network servers in the corporate OT network for performing cryptographic tasks. Deploying dedicated servers for this task in the corporate OT network brings several benefits:

1. *Security.* The Edge Server's network interface connected to the corporate DMZ results in an increased attack surface. Storing unique keys for each IIoT device on such an exposed system is neither scalable nor acceptable from a security perspective. Therefore, it is recommended to store the key material on a CKMD, constituting a separate, shielded component.

2. *Physical key access.* The CKMD Server offers a restricting interface that physically prohibits access to the cryptographic key material. Keys are encapsulated within the CKMD and well-protected against extraction attempts. Moreover, decryption operations are restricted to legitimate operations. In particular, decryption requests

are rejected whenever authenticity verification fails. This restriction is an additional hurdle in abusing the CKMD.

3. *Performance considerations.* Decryption and verification of data entries from hundreds or thousands of IIoT devices at potentially high rate can result in performance bottlenecks. The outsourcing of these operations to the CKMD Server(s) relieves the Edge Server from this effort and supports load balancing if needed.

Following successful decryption, the Edge Server forwards the decrypted IIoT data to the destination Application Server for further processing.

Due to the sensitivity of transmitted data, communication between Edge Server and both CKMD Server and Application Server is end-to-end protected, for instance using TLS transport-layer encryption. Furthermore, an IDS monitors the network traffic in the corporate OT network and in the DMZ, identifying anomalous behavior and raising alarms on suspicious traffic. However, we assume that because of security reasons the IDS neither has access to key material nor can it intercept TLS connections within the corporate OT network.

Relevance of Subliminal Communication

One main observation concerning Figure 6.6 is that the presented architecture can effectively protect against network-layer or transport-layer covert channels or subliminal channels. Data concentration and potentially distinct protocols in the sensor network and over the Internet – for instance IPv6 and UDP addressing in the sensor network vs. IPv4 and TLS over TCP in the Internet – can effectively block all attempts for covert communication. Assuming a compromise of IIoT sensors and the Edge Server, the proposed GCM subliminal channel is an ideal candidate for hidden communication between these two. The monitoring IDS in Figure 6.6 cannot unveil this subliminal channel unless the observed encrypted traffic pattern differs from the benign case.

A second observation concerns the assumption of CKMDs supporting the IV to be a user-provided parameter for encryption: IIoT device manufacturers face tremendous pressure to decrease the costs per unit due to the high production volume. The production volume of CKMDs can be increased (and the cost per module can be reduced) by implementing stateless operation and flexible options to configure cryptographic input parameters. On the IIoT device itself, on the other hand, state keeping usually is necessary also for accomplishing other tasks. Hence, computation of IVs on the IIoT device does not add cost. Therefore, we argue that our assumption of stateless CKMD operation, allowing applications to provide the IV for GCM encryption, is absolutely legitimate.

As a final note, we remark that a straightforward solution seems to be the implementation of an end-to-end TLS secured connection between IIoT devices and Application Server. However, complex security architectures like the one shown in Figure 6.6 are a prerequisite to operate the huge base of existing IIoT devices. The available capacity of typical sensor

networks, as well as the huge count and limited computational performance of IIoT devices, are commonly prohibitive factors in implementing end-to-end TLS.

6.2.6 Mitigations

To avoid the attack scenarios highlighted above, a natural solution is to use a different mode of operation instead. For example, the use of the recent GCM-SIV scheme can be recommended. GCM-SIV retains most attractive properties and, in particular, the performance benefits of GCM, but implements means to alleviate issues arising from IV reuse by choosing the IV for counter mode encryption depending on the message itself. AES-GCM-SIV [106], which has been standardized in [103], enhances the construction further to obtain improved security bounds. Compared to GCM, GCM-SIV and AES-GCM-SIV have the only drawback of no longer operating in an online fashion, i.e. requiring two passes through the data for encryption instead of just one.

Alternatively, an operational mode that does not deploy counter mode encryption can be used instead like, for example, the traditional choice of Cipher Block Chaining (CBC) [3] in conjunction with a Hash-based Message Authentication Code (HMAC). While the reuse of IVs downgrades security also for modes like GCM-SIV or CBC, security breaches are not as detrimental as for GCM. In particular, it is no longer possible to achieve decryption by issuing an encryption request, ruling out the possibilities for subliminal channels described in this chapter.

However, AES-GCM is a very popular cipher. Modes like CBC might be difficult to deploy due to their downsides with respect to encryption speed. If compatibility with existing implementations is required, the use of GCM might be the only option in practical scenarios. Thus, the question arises how to improve security of architectures described in this chapter while retaining GCM encryption.

As described above, a universal method for avoiding attack possibilities is letting the CKMD generate an IV for each encrypted message. Since we assume the CKMD stateless in most cases, however, IV generation might have to take place randomly, allowing the random IV to be used as a subliminal channel originating from the CKMD. The use of random IVs would, furthermore, incur the drawbacks described in Section 6.2.2. Since random IV generation requires transmitting the IV values with each message, it might not only require a modification of existing implementations, but also a protocol adaptation.

A further simple method to avoid most attacks described in Section 6.2.3 is to ensure that forward and reverse direction use different keys or IVs. Hence, from a given configured key, subkeys can be derived for each direction, respectively. This approach allows preventing all attacks that are based on performing decryption by issuing an encryption request or vice versa. It does not prevent the “forbidden attack”, however, which allows performing authenticating encryption of an arbitrary amount of messages by requesting just two encryptions with the same IV.

If the same encryption key has to be used in both directions for reasons of, e.g., compatibility with existing implementations, a further possibility is to preconfigure certain parts

of the IVs on the CKMD. It is sufficient if the CKMDs of the communicating parties require one bit of the IV to be set to respective different values for both directions for the subliminal channel, as described in this chapter, to stop working. In fact, NIST recommendation [77] already specifies that a part of the IV identifies the device and, hence, is constant for all invocations on one CKMD. Therefore, this recommendation would only have to be enforced by the CKMD. However, just as using different encryption keys for communication directions, the approach similarly fails to protect against message forgery from two authenticated messages sharing one IV.

6.2.7 Discussion

Despite being understood as reasonable approaches for achieving security when being used on their own, combining GCM and CKMDs risks to yield a false sense of security. We highlighted attack possibilities that can be used to circumvent restrictions that might be applied by a CKMD in performing encryption or decryption tasks, and showed in particular how these methods can be used to establish a subliminal channel utilizing the authentication tag. Since the CKMD would deny decryption, the establishment of this subliminal channel would not be possible otherwise. For an attacker, the subliminal channel has very attractive properties and allows communicating substantial amounts of hidden information in infrastructures that aim to meet highest demands in terms of security and privacy. Various scenarios allow clandestinely transmitting information. While we focused on GCM, the technique is also applicable to encryption modes that are constructed in a similar way.

We went into detail for an exemplary infrastructure in the field of IIoT and additionally highlighted several approaches for how the threats discussed here might be mitigated. The most potent remedy is to choose a different operational mode like GCM-SIV for encryption. Alternatively, IVs can be chosen directly on the CKMD to avoid attack possibilities, which, however, is likely to enable a subliminal channel originating from the CKMD.

Since the described threats target specifically high-security infrastructures, future development of operational modes and future architectural design of high-security infrastructures should consider possibilities for hidden communication using approaches we described.

CHAPTER 7

Discussion

In the following sections, we will provide a summarizing discussion of the work we have conducted throughout the previous chapters. To show relevance for deployment in practical applications, we will discuss implications for the construction of real-world IDSs of our rather research-oriented experiments and our theoretical work.

As a basis for future work, we also provide pointers for possible research questions that still need investigation, but we also discuss challenges that must be considered when working in this area.

7.1 Summarizing Discussion

Our goal in this thesis was to investigate several aspects of ML-based IDS design that are highly relevant when deploying such techniques in a highly security-critical infrastructure.

We investigated the use of unsupervised methods with focus on detection performance, processing speed and interpretability. This work on streaming outlier detection algorithms has shown that no ideal method is available to date that matches the characteristics and challenges of processing network traffic. Besides the problem of processing speed, which we tried to alleviate with our dSalmon framework, many modern approaches for outlier detection rely on workings that do not allow interpretation of the outlier score. In addition, our experiments have shown that also considering detection performance unsupervised ML is not yet able to match the good results that can usually be obtained with supervised ML. Approaches like the OptOut feature vector we introduced in Section 3.3.4 might be evaluated to optimize outlier detection performance. However, such specialized feature vectors have to stand the test of time, as we cannot preclude that the OptOut feature vector delivers good performance only for the attacks or the specific data we have used in our experiments.

Still, the concept of anomaly detection yields potential for being used in NID, since only unsupervised ML allows detecting unknown attacks. If detection performance is not ideal, interpretability becomes even more important to be able to manually investigate observed data and assess whether specific flows might constitute an attack. It is therefore necessary to provide as much context of an observed flow to human security experts as possible. An interesting option is to additionally capture when similar flows have been observed and with which periodicity they reoccur. Our approach to achieve this goal can thus yield substantial potential for practical deployability of IDSs, as this temporal information in many cases is easier interpretable than a feature vector consisting of statistical features. Furthermore, in contrast to many other outlier detection algorithms for streaming data, our SDOstream is lightweight enough to allow usage in this domain, even when requiring substantial memory lengths.

Apart from unsupervised ML, more traditional supervised ML must of course not be disregarded for IDSs in security-critical infrastructures. Considering detection performance of such methods, the related research has demonstrated remarkable results, so in this thesis we filled the gap of augmenting the most promising techniques with methods for providing explainability. Comparing both approaches, we come to the conclusion that methods that provide interpretability by themselves have to be favored compared to using additional methods, like PDPs, that explain results from ML models lacking inherent explainability. PDPs, ALEs and sequential PDPs were able to depict to a certain extent how feature values influence the classifiers' predictions, but did not convince entirely in defending against adversarial ML. Still, if NNs or RFs are used, they can provide valuable information about the classification process.

7.2 Challenges and Difficulties

The field of ML has shown many challenges and pitfalls and requires a high level of precision when conducting experiments. In this section, we briefly outline difficulties we encountered during our work on this thesis.

7.2.1 Research Challenges in Our Field

Conducting high-quality research in the field of NID involves a few challenges, which we show up in this section.

One of the biggest challenges is obtaining suitable data for training and evaluating IDSs. Curating data that depict human behavior for a representative amount of time (e.g. months) raises substantial data protection concerns and prompts huge storage and transfer needs, which explains why only very few datasets of such kind are publicly available. The problem becomes even more difficult, however, if labeled data are required, since labelling is an extremely expensive, time-consuming and error-prone process. In bilateral projects with industry, real datasets can be used for investigating our algorithms. However, publication of data from project partners is often not possible and therefore publication

of results based on such data is difficult and also hinders the fulfillment of reproducibility standards. In this thesis, we often used the CIC-IDS-2017 and UNSW-NB15 datasets, which met our requirements, but in previous research showed some shortcomings [80]. Not only would our own demand for high-quality experimentation request the use of more datasets, but we noticed that this issue also frequently raises criticism in the scientific community. We were unable to overcome this issue due to the lack of other appropriate datasets.

An issue related to the availability of datasets is the question of which metrics to use for IDS evaluation. As described in Chapter 2, there is no natural metric for how performance of an IDS has to be evaluated. We therefore computed several metrics to provide an overview over how methods perform. However, since different authors provided different metrics in their publications, results are not always comparable to previous work and results from existing work cannot always be compared. It should also be noted that providing several metrics still leaves some problems, since for some tasks a single metric is needed. Examples include hyperparameter search or certain feature importance methods. We tried to select the most usual or most reasonable metric in such cases.

A challenge that is very specific to methods we explored in this paper involves the computational demands of streaming data processing. If a model should be updated with every observed sample, batch data processing usually is not possible, making implementations in Python or other scripting languages commonly used in data science exceedingly slow. We tried to alleviate this problem with our dSalmon framework, also allowing future researchers to process data streams with good speed. The problem remains, however, that new algorithms or algorithm adaptations require implementation in a relatively low-level programming language and manually making use of multithreading possibilities. While the efficient implementation of completed algorithms is no big problem in practice, this provides an obstacle for many data science researchers or slows down development.

7.2.2 Unanticipated Difficulties

As already mentioned above, the search for good datasets is a difficult task in many cases. We noticed that in particular for analyzing temporal patterns the search for a suitable dataset was very difficult. The reason for this problem can be traced back to the fact that in many cases datasets are created for a short timespan only. However, for our scenario we required patterns that are stable for a longer period, so that periodicities in the range of, e.g., days can be detected. The reason for why datasets representing a longer period are rarely available is likely due to the substantial storage and transfer needs for datasets of this size. We note that this problem is dominant in the research phase, but less in the application phase, since during application the algorithm can simply be applied to live data as it is captured.

The task of traffic separation turned out to be more difficult than we originally assumed. In fact, the usual approach to assess anomalousness with NNs is to use an autoencoder.

Preliminary experiments to use autoencoder architectures for this task did not succeed. We furthermore attempted to perform flow separation by using gradient descent on a differentiable total anomalousness instead of using a Viterbi-like algorithm, which similarly did not yield good results. Unfortunately, the usual publication processes in our field, which enforce length-constrained publications, did not allow us to describe these failed attempts in our published papers.

7.3 Opportunities for Improvement and Future Research

To date, the performance results that can be obtained by unsupervised ML techniques are not yet satisfactory. Hence, it is still an important research direction to evaluate new approaches for anomaly detection or test new feature vectors or detectability of certain attack types. Our proposed methods have shown new directions that have benefits in particular when interpretability is important. Due to the vast amount of possibilities to construct anomaly detection algorithms, a targeted development of an ideal outlier detection algorithm for application in the field of IDS is hardly possible, but rather involves a certain amount of trial-error.

As already mentioned above, an issue that is omnipresent in the field of ML-based IDS research is that of obtaining good datasets. We therefore consider the creation or consolidation of new datasets a pressing problem, which deserves more attention in the scientific community. Since it is almost unavoidable that minor mistakes or inaccuracies lead to a dataset that might later be disputed by independent reviewers, we consider it more meaningful to provide scripts and frameworks for dataset generation in addition to the datasets themselves, so that good datasets can evolve in the course of time. This approach also alleviates the problem of being limited to certain attacks, since new attacks can then easily be added.

A particular area of potential future research applies to our work on the subliminal channel exploiting GCM. Our work in this area contrasts previous work in this area by being based on a particular architecture that is highly relevant for high-security infrastructures. Hence, future work might explore both, the use of further encryption methods in such a scenario but also possible means of defense.

For our work on separating flows in encrypted tunnel traffic, a similar observation can be made. Also in this case the problem setting that we explore is very novel. Future work might explore different neural network architectures for assessing whether individual packets are anomalous. Alternatively, different methods for performing the separation, i.e. minimizing total anomalousness, might be investigated and optimized. Furthermore, an interesting question would be further investigating which types of traffic show patterns that are sufficiently distinctive to allow separation and whether this only holds when mixing it with very different traffic or also with traffic of the same kind. Also implications for practice and defense strategies could further be explored.

7.4 Recommendations for IDS Construction

Highly application-oriented scenarios motivated the problem setting we explore in this thesis. To match this highly practical setting, we want to provide some practical guidelines from the results of our work.

Our experiments have shown that neither unsupervised ML in terms of anomaly detection nor supervised ML are ideally suited for attack detection in NID. In particular, while unsupervised techniques show detection performances that are hardly sufficient in many scenarios, supervised techniques can only detect known attack types and explainability is not as straightforward as when using simple nearest-neighbor-based outlier detection. It therefore appears reasonable to deploy a system combining both paradigms. To the effect of interpretability, it then makes sense to provide the results of both techniques to the data analyst instead of just providing an anomaly score that is formed in an abstract way. Furthermore, explainability should be kept in mind in the design of both unsupervised and supervised techniques, e.g., by relying on techniques as outlined in this thesis.

An aspect that is often overlooked in IDS design is the expressiveness of the temporal reoccurrence of specific types of observed traffic for understanding the type of traffic and aiding NID. Indeed, due to strongly time-dependent human behavior, certain types of traffic are likely to be seen at specific points in time and the information at what points in time this particular type of traffic is usually seen may provide much more information than a feature vector obtained from abstract statistics. It is therefore reasonable to design ML methods to provide this kind of information.

We consider the technique of flow identification in encrypted tunnel traffic we introduced in this thesis a particular interesting approach. The basic experiments that we conducted in this thesis clearly showed that the risk of an attacker being able to successfully identifying individual flows cannot be neglected, even though our method or our deep learning model doubtlessly still could be improved. Hence, to improve security additional techniques for obfuscation of traffic characteristics should be used for tunnel encryption that avoid leaking observable traffic patterns to eavesdroppers even though such techniques harm performance slightly. On the defender's side, the same technique certainly shows up an interesting direction, but considering our conducted experiments does not yet seem to be mature enough for being used to analyze encrypted tunnel traffic.

Finally, our experiments underlined an assertion that should be evident from common sense but might be neglected in everyday life when resources for implementing security are sparse: While an IDS is a reasonable concept to improve network security, it cannot be the only line of defense. This, however, holds true for any other security technique as well. We have made this fact very clear by pointing out the possibility for malware communication that cannot possibly be detected by IDSs. In addition to the use of an IDS, additional measures should thus be taken to ensure that malware does not even enter the enterprise network, which, e.g., might involve training employees in security-related topics.

CHAPTER 8

Conclusion

In the light of modern high-security infrastructures, research in network security becomes more and more important. In such scenarios, challenges must be considered that go beyond just achieving high accuracy in attack detection by far. Explainability is a major concern and increasing attack sophistication requires detectors able to identify targeted attacks of still unknown nature.

In this thesis, we have covered several aspects of security in such environments. The area of outlier detection in streaming data receives its significance in this respect from the fact that only unsupervised methods allow to detect unknown attacks. Our overview and our algorithm design and implementations substantially contribute to the available state of research, allowing researchers to use the most suitable method for their intended task. We have investigated explainability and interpretability in both supervised and unsupervised domains and devised an algorithm for unsupervised anomaly detection that is particularly well suited for the characteristics and challenges in NID. Due to the characteristics of network data, our focus naturally was on the online processing of streaming data.

Our work has revealed that various pressing research problems remain in the field of NID research. Not only do detection methods themselves still need to be improved, but also the realistic evaluation of IDSs provides many challenges to researchers. Considering detection accuracy, research is only at the beginning for unsupervised attack detection and substantial further research is required on fruitfully applying unsupervised methods on network traffic. We have additionally introduced several novel ideas that open up new research directions, namely our described attack scenario in the GCM subliminal channel, our approach of encrypted traffic analysis and our novel approach to handle temporal patterns. All these concepts prompt the evaluation of a wide range of further methods in the described scenarios.

Considering our research questions, we note that the construction of potent IDSs and the secure use of encryption still pose challenges in high-security infrastructures. We asked whether it is possible to detect network attacks with good accuracy with unsupervised methods. Unfortunately, based on our experiments we cannot attest good detection accuracy to most anomaly detection methods. Additionally, a kNN approach, which would provide explainable results, also is among the slowest anomaly detectors. This downside, however, we have been able to address by designing an anomaly detector that provides both, interpretability and decent runtimes. In the supervised domain, results are more promising, since predictions from well-performing supervised detectors can be made explainable to a certain extent.

We can derive several lessons learned and generalized findings from the work in this thesis. In general, at least in the case of infrastructures where high security has to be achieved, simple methods like, e.g., shallow models or kNN-based approaches, seem to be a better choice than modern methods like deep learning. While a substantial body of research has been conducted in the recent years on explainability, such methods are a partial solution, but no perfect solution to explain the decision processes in complex classifiers, e.g., when it comes to investigating regions that might be exploited by adversarial samples. Even though best practices in such infrastructures involve the use of reasoned and well-tested security techniques, our work on GCM and tunnel encryption has shown that shortcomings and security risks might still exist that have not been discovered or anticipated before.

List of Figures

List of Tables

List of Algorithms

Acronyms

kNN k Nearest Neighbors. 18, 19, 28, 29, 40, 48–50, 52, 55, 56, 60, 63, 68, 78, 150

AAP Adjusted Average Precision. 3, 15, 55, 56, 80, 95

AEAD Authenticated Encryption with Associated Data. 125, 126, 128, 133

AES Advanced Encryption Standard. 124, 125, 129, 130, 132, 138, 141

ALE Accumulated Local Effects. 4, 6, 20, 21, 108–110, 144, 152

AML Adversarial Machine Learning. 111, 113, 114

AP Average Precision. 15, 55, 56

AP@n Adjusted Average Precision at n. 3, 15, 55, 56, 80, 95

APT Advanced Persistent Threat. 22

ARI Adjusted Rand Index. 98, 99, 101

ASA Algorithm Substitution Attack. 125

CBC Cipher Block Chaining. 141

CKMD Cryptographic Key Management Device. 125–128, 131–142, 152

CMS Count-min sketch. 45

CTR Counter Mode. 126, 132

CW Carlini-Wagner method. 113

DDoS Distributed Denial-of-Service. 46

DMZ Demilitarized Zone. 139, 140

DoS Denial-of-Service. 118

DPI Deep Packet Inspection. 9

DT Decision Tree. 15, 16, 20, 106

EVP Envelope. 136

EWMA Exponentially Weighted Moving Average. 28, 29, 32, 60, 65, 69, 71, 72, 77

FT Fourier Transform. 65, 69, 72–74, 76, 77, 82, 84, 152

GCM Galois/Counter Mode. 6, 7, 124–130, 132, 133, 135, 138, 140–142, 146, 149, 150, 152

HBOS Histogram-based Outlier Score. 18, 48–50, 52, 54–57

HMAC Hash-based Message Authentication Code. 141

IAT Inter-Arrival Time. 10, 27, 62, 66, 75, 85, 88–91, 95, 98, 101, 107, 112–114, 118, 121, 152

IDS Intrusion Detection System. 2–4, 6, 7, 11–14, 18, 19, 21–23, 25, 26, 46–48, 59, 85, 105–108, 110–112, 119, 121, 122, 126, 140, 143–147, 149, 150

iForest Isolation Forest. 31, 48–50, 52, 55, 56

IIoT Industrial Internet of Things. 127, 138–142, 152

IT Information Technology. 1, 2, 7

IV Initialization Vector. 125–127, 129–133, 135, 136, 140–142

LOF Local Outlier Factor. 18, 31, 48–50, 52, 55, 56, 63, 68

MAC Message Authentication Code. 124

ML Machine Learning. vii, 2–7, 9–16, 19–21, 33, 47, 59–61, 70, 85–88, 94, 105–108, 110–112, 115, 122, 143, 144, 146, 147

MLP Multilayer Perceptron. 17, 107–110, 112, 113, 152, 153

MuI Model under Investigation. 20, 108–110

NAT Network Address Translation. 136

NID Network Intrusion Detection. 1–5, 12, 13, 16, 17, 23, 46, 57, 70, 85–87, 94, 95, 106, 144, 147, 149, 153

NN Neural Network. 12, 15–18, 20, 62, 92–95, 98, 99, 102, 105, 106, 112–116, 119, 122, 144, 145, 151

NTA Network Traffic Analysis. 3, 7, 9–11, 47, 51, 59, 63, 153

OT Operating Technology. 139, 140

P@n Precision at n. 15, 36, 55, 56

PCAP Packet Capture file. 9, 52

PDP Partial Dependence Plot. 4, 6, 20, 21, 108–110, 112, 119–122, 144, 152

PGD Projected Gradient Descent. 114

RAPL Running Average Power Limit. 38

RBG Random Bit Generator. 130

ReLU Rectified Linear Unit. 108

RF Random Forest. 15, 16, 20, 64, 106–110, 144, 152

RNN Recurrent Neural Network. 4, 7, 18, 106, 111–114, 116, 119, 121, 122

ROC Receiver Operating Characteristic. 15, 80

ROC-AUC Area under the ROC curve. 3, 15, 36, 40, 52, 54–56, 68, 79, 80, 95, 153

RRCF Robust Random Cut Forest. 28, 31, 42

RW Reference Window. 27, 28, 32, 42

SDO Sparse Data Observers. 48–50, 52, 54–57, 60, 64, 65, 67, 68, 70, 153

SETUP Secretly Embedded Trapdoor with Universal Protection. 125

SIDS Signature-based Intrusion Detection System. 11, 12

SMD Simple Matching Distance. 29

SW Sliding Window. 19, 27–32, 40, 42, 63, 64, 77, 151

TPM Trusted Platform Module. 127

TTL Time-to-Live. 55, 107–110, 112

VPN Virtual Private Network. 3, 5, 60, 62, 86–90, 101

XOR Exclusive Or. 129, 131, 132, 134, 135, 138

Bibliography

[1] What is critical infrastructure? why does critical infrastructure security matter? URL https://www.paloaltonetworks.com/cyberpedia/what-is-critical-infrastructure. Accessed: 2023-05-07.

[2] Nielsen's law of internet bandwidth. URL https://www.nngroup.com/articles/law-of-bandwidth/. Accessed: 2023-05-07.

[3] Fips 81, des modes of operation. *Federal Information Processing Standards Publication 81*, page 17, 1980.

[4] Kdd cup 1999 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999. Accessed: 2023-05-07.

[5] Supplement to part 11: Wireless medium access control (mac) and physical layer (phy) specifications: High speed physical layer in the 5 ghz band. IEEE Std. 802.11a-1999, 1999.

[6] Supplement to part 11: Wireless medium access control (mac) and physical layer (phy) specifications: Higher speed physical layer (phy) extension in the 2.4 ghz band. IEEE Std. 802.11b-1999, 1999.

[7] Fips 197, Advanced encryption standard (AES). *Federal Information Processing Standards Publication 197*, page 51, 2001.

[8] Supplement to part 11: Wireless medium access control (mac) and physical layer (phy) specifications: Further higher data rate extension in the 2.4 ghz band. IEEE Std. 802.11g-2003, 2003.

[9] Supplement to part 11: Wireless medium access control (mac) and physical layer (phy) specifications: Medium access control (mac) security enhancements. IEEE Std. 802.11i-2004, 2004.

[10] Supplement to part 11: Wireless lan medium access control (mac)and physical layer (phy) specifications: Enhancements for higher throughput. IEEE Std. 802.11n-2009, 2009.

[11] Supplement to part 11: Wireless medium access control (mac) and physical layer (phy) specifications: Enhancements for very high throughput for operation in bands below 6 ghz. IEEE Std. 802.11ac-2013, 2013.

[12] RFC 7011 - Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. Technical report, Internet Engineering Task Force (IETF), September 2013. URL https://www.ietf.org/rfc/rfc7011.txt.

[13] UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013. Accessed: 2023-05-07.

[14] Ieee standard for local and metropolitan area networks-media access control (mac) security. *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pages 1–239, 2018. doi: 10.1109/IEEESTD.2018.8585421.

[15] Wireshark wiki: WLAN, 2020. URL https://wiki.wireshark.org/CaptureSetup/WLAN. Accessed: 2023-05-07.

[16] Mayank Agarwal, Sanketh Purwar, Santosh Biswas, and Sukumar Nandi. Intrusion detection system for PS-Poll DoS attack in 802.11 networks using real time discrete event system. *IEEE/CAA Journal of Automatica Sinica*, 4(4):792–808, 2017. ISSN 2329-9274. Conference Name: IEEE/CAA Journal of Automatica Sinica.

[17] Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52, 2005.

[18] Azim Ahmadzadeh and Berkay Aydin. Multivariate Timeseries Feature Extraction on SWAN Data Benchmark (SWAN_Features), 2020. URL https://github.com/charlespwd/project-title. GSU Data Mining Lab, Bitbucket repository.

[19] Open Charge Alliance. Open charge point protocol. https://www.openchargealliance.org/protocols/, 2015. Accessed: 2023-05-07.

[20] Wi-Fi Alliance. WPA3 Specification, 2020. Version 3.0.

[21] Mashael AlSabah and Ian Goldberg. Performance and security improvements for tor: A survey. *ACM Computing Surveys (CSUR)*, 49(2):1–36, 2016.

[22] Riyad Alshammari and A Nur Zincir-Heywood. A flow based approach for ssh traffic detection. In *2007 IEEE int. conf. on systems, man and cybernetics*, pages 296–301. IEEE, 2007.

[23] Riyad Alshammari and A Nur Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying ssh and skype. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–8. IEEE, 2009.

[24] Riyad Alshammari and A Nur Zincir-Heywood. A preliminary performance comparison of two feature sets for encrypted traffic classification. In *Proc. of the Int. Workshop on Computational Intelligence in Security for Information Systems CISIS'08*, pages 203–210. Springer, 2009.

[25] Riyad Alshammari and A Nur Zincir-Heywood. An investigation on the identification of voip traffic: Case study on gtalk and skype. In *2010 Int. Conf. on Network and Service Management*, pages 310–313. IEEE, 2010.

[26] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? *Computer networks*, 55(6):1326–1350, 2011.

[27] Riyad Alshammari, Peter I Lichodzijewski, Malcolm Heywood, and A Nur Zincir-Heywood. Classifying ssh encrypted traffic with minimum packet header features using genetic programming. In *Proc. of the 11th Annual Conf. Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2539–2546, 2009.

[28] Sara A Althubiti, Eric Marcell Jones, and Kaushik Roy. Lstm for anomaly-based network intrusion detection. In *2018 28th International telecommunication networks and applications conference (ITNAC)*, pages 1–3. IEEE, 2018.

[29] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and information systems*, 51(2):339–367, 2017.

[30] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In *Proc. of the 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1723–1732, 2017. ISBN 978-1-4503-4887-4.

[31] Blake Anderson and David A. McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2016, Vienna, Austria, October 28, 2016*, pages 35–46, 2016.

[32] F. Angiulli and F. Fassetti. Detecting distance-based outliers in streams of data. In *Proc. of the 16th ACM Conf. on Information and Knowledge Management*, CIKM'07, pages 811–820, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-803-9. doi: 10.1145/1321440.1321552.

[33] Rafal A. Angryk, Petrus C. Martens, Berkay Aydin, Dustin Kempton, Sushant S. Mahajan, Sunitha Basodi, Azim Ahmadzadeh, Xumin Cai, Soukaina Filali Boubrahimi, Shah Muhammad Hamdi, Michael A. Schuh, and Manolis K. Georgoulis. Multivariate time series dataset for space weather data analytics. *Scientific Data*, 7(227), 2020.

[34] Daniel W. Apley and Jingyu Zhu. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *arXiv:1612.08468 [stat]*, December 2016. arXiv: 1612.08468.

[35] Jacob Appelbaum, Marsh Ray, Karl Koscher, and Ian Finder. vpwns: Virtual pwned networks. In *2nd USENIX Workshop on Free and Open Communications on the Internet. USENIX Association*, 2012.

[36] Marcel Armour and Bertram Poettering. Subverting decryption in AEAD. Technical Report 987, 2019. URL `http://eprint.iacr.org/2019/987`.

[37] Daniel J Arndt and A Nur Zincir-Heywood. A comparison of three machine learning techniques for encrypted network traffic analysis. In *2011 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 107–114. IEEE, 2011.

[38] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, pages 1–7, 1999.

[39] Md Ahsan Ayub, William A Johnson, Douglas A Talbert, and Ambareen Siraj. Model evasion attack on intrusion detection systems using adversarial machine learning. In *2020 54th annual conference on information sciences and systems (CISS)*, pages 1–6. IEEE, 2020.

[40] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k-means clustering via lightweight coresets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1119–1127, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520.

[41] Maximilian Bachl. *Machine learning methods for communication networks: Characterization and analysis of selected use cases.* PhD thesis, TU Wien, 2021.

[42] Maximilian Bachl, Alexander Hartl, Joachim Fabini, and Tanja Zseby. Walling Up Backdoors in Intrusion Detection Systems. In *Big-DAMA '19*, pages 8–13, Orlando, FL, USA, 2019. ACM.

[43] Lamiaa Basyoni, Noora Fetais, Aiman Erbad, Amr Mohamed, and Mohsen Guizani. Traffic analysis attacks on tor: a survey. In *2020 IEEE Int. Conf. on Informatics, IoT, and Enabling Technologies (ICIoT)*, pages 183–188. IEEE, 2020.

[44] David M. Beazley. SWIG: An easy to use tool for integrating scripting languages with C and C++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, page 15, USA, 1996. USENIX Association.

[45] M. Bellare, P. Rogaway, and D. Wagner. EAX: A conventional authenticated-encryption mode. Technical Report 069, 2003. URL `https://eprint.iacr.org/2003/069`.

[46] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In *Advances in Cryptology – CRYPTO 2016*, pages 247–276, Berlin, Heidelberg, 2016. Springer. ISBN 978-3-662-53018-4. doi: 10.1007/978-3-662-53018-4_10.

[47] Mihir Bellare, Kenneth Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. Technical Report 438, 2014. URL `http://eprint.iacr.org/2014/438`.

[48] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. Technical Report 808, 2015. URL `http://eprint.iacr.org/2015/808`.

[49] M. Bhuyan, D. Bhattacharyya, and J. Kalita. Network anomaly detection: Methods, systems and tools. *Communications Surveys Tutorials, IEEE*, PP(99):1–34, 2013.

[50] M. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. A multi-step outlier-based anomaly detection approach to network-wide traffic. *Information Sci.*, 348:243–271, 2016.

[51] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[52] Battista Biggio, Igino Corona, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. Bagging classifiers for fighting poisoning attacks in adversarial environments. In *10th Int'l Workshop on Multiple Classifier Systems, volume 6713 of LNCS*, pages 350–359, Naples, Italy, 2011. Springer.

[53] Jens-Matthias Bohli and Rainer Steinwandt. On subliminal channels in deterministic signature schemes. In *International Conference on Information Security and Cryptology*, pages 182–194. Springer, 2004.

[54] Jens-Matthias Bohli and Rainer Steinwandt. On subliminal channels in deterministic signature schemes. In *Information Security and Cryptology – ICISC 2004*, pages 182–194, Berlin, Heidelberg, 2005. Springer. ISBN 978-3-540-32083-8. doi: 10.1007/11496618_14.

[55] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. A subliminal-free variant of ecdsa. In *International Workshop on Information Hiding*, pages 375–387. Springer, 2006.

[56] Jens-Matthias Bohli, María Isabel González Vasco, and Rainer Steinwandt. A subliminal-free variant of ECDSA. In *Information Hiding*, pages 375–387, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-74124-4. doi: 10.1007/978-3-540-74124-4_25.

[57] Kendrick Boyd, Kevin H. Eng, and C. David Page. Area under the precision-recall curve: Point estimates and confidence intervals. In *Machine Learning and Knowledge Discovery in Databases*, pages 451–466, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-40994-3.

[58] M. M. Breunig, H.-P. Kriegel, et al. LOF: identifying density-based local outliers. In *Proc. of the 2000 ACM SIGMOD Int. Conf. on Management of data*, pages 93–104, 2000.

[59] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, Secondquarter 2016.

[60] Thanh Bui, Siddharth Rao, Markku Antikainen, and Tuomas Aura. Client-side vulnerabilities in commercial vpns. In *Nordic Conference on Secure IT Systems*, pages 103–119. Springer, 2019.

[61] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS. Technical Report 475, 2016. URL `http://eprint.iacr.org/2016/475`.

[62] CAIDA. The CAIDA UCSD network telescope "patch tuesday" dataset. `http://www.caida.org/data/passive/telescope-patch-tuesday_dataset.xml`. Accessed: 2023-05-07.

[63] G. O. Campos, A. Zimek, et al. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, 2016. ISSN 1573-756X. doi: 10.1007/s10618-015-0444-8.

[64] Zigang Cao, Gang Xiong, Yong Zhao, Zhenzhen Li, and Li Guo. A survey on encrypted traffic classification. In *Int. Conf. on Applications and Techniques in Information Security*, pages 73–81. Springer, 2014.

[65] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *S&P*, pages 39–57. IEEE, May 2017.

[66] Chun-Hao Chang, Ladislav Rampasek, and Anna Goldenberg. Dropout feature ranking for deep learning models. *arXiv:1712.08645*, 2017.

[67] Chris Chatfield. *The analysis of time series: An introduction*. Chapman and Hall/CRC, London, UK, 2003.

[68] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust Decision Trees Against Adversarial Examples. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1122–1131, Long Beach, CA, 2019. PMLR.

[69] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the WIDE project. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.

[70] P. Ciaccia, M. Patella, et al. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the 23rd VLDB conference*, pages 426–435, 1997.

[71] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[72] Nick Craswell. *Precision at n*, pages 2127–2128. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9.

[73] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. Rapl: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194. IEEE, 2010.

[74] Amit Dhurandhar, Pin-Yu Chen, Karthikeyan Shanmugam, Tejaswini Pedapati, Avinash Balakrishnan, and Ruchir Puri. Model Agnostic Contrastive Explanations for Machine Learning Classification Models. 2018.

[75] Qingkuan Dong and Guozhen Xiao. A subliminal-free variant of ecdsa using interactive protocol. In *2010 International Conference on E-Product E-Service and E-Entertainment*, pages 1–3. IEEE, 2010.

[76] Zakir Durumeric, Michael Bailey, and J. Alex Halderman. An internet-wide view of internet-wide scanning. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 65–78, Berkeley, CA, USA, 2014. USENIX Association.

[77] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/-counter mode (GCM) and GMAC. Technical Report NIST Special Publication (SP) 800-38D, National Institute of Standards and Technology, November 2007. URL `https://csrc.nist.gov/publications/detail/sp/800-38d/final`.

[78] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[79] Gregory G Enas and Sung C Choi. Choice of the smoothing parameter and efficiency of k-nearest neighbor classification. *Computers & Mathematics with Applications*, 12(2, Part A):235–244, 1986.

[80] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12. IEEE, 2021.

[81] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, May 1997.

[82] Joachim Fabini, Tanja Zseby, and Michael Hirschbichler. Representative delay measurements (rdm): Facing the challenge of modern networks. *EAI Endorsed Transactions on Creative Technologies*, 2(6):e5, 2015.

[83] C. Fachkha, E. Bou-Harb, and M. Debbabi. Towards a forecasting model for distributed denial of service activities. In *2013 IEEE 12th International Symposium on Network Computing and Applications*, pages 110–117, Aug 2013.

[84] Niels Ferguson. Authentication weaknesses in GCM. *Comments submitted to NIST Modes of Operation Process*, pages 1–19, 2005.

[85] Daniel C. Ferreira, Félix Iglesias Vázquez, Gernot Vormayr, Maximilian Bachl, and Tanja Zseby. A meta-analysis approach for feature selection in network traffic research. In *Proc. of the Reproducibility Workshop*, Reproducibility '17, pages 17–20. ACM, 2017. ISBN 978-1-4503-5060-0.

[86] Daniel C. Ferreira, Maximilian Bachl, Gernot Vormayr, Félix Iglesias, and Tanja Zseby. Curated research on network traffic analysis, November 2018.

[87] Alejandro Flores-Velazco and David M. Mount. Coresets for the nearest-neighbor rule. In *28th Annual European Symposium on Algorithms (ESA 2020)*, Dagstuhl, Germany, 2020. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[88] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 1–24, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-45537-0.

[89] P. Fournier-Viger. An introduction to periodic pattern mining, July 2016. URL `http://data-mining.philippe-fournier-viger.com/an-introduction-to-the-discovery-of-periodic-patterns-in-data/`.

[90] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[91] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, RFC Editor, February 2011. URL `http://www.rfc-editor.org/rfc/rfc6071`. Num Pages: 63.

[92] Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[93] Herman Galteland and Kristian Gjøsteen. Subliminal channels in post-quantum digital signature schemes. *Cryptology ePrint Archive*, 2019.

[94] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

[95] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1):18 – 28, 2009.

[96] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. An evaluation framework for intrusion detection dataset. In *International Conference on Information Science and Security (ICISS)*, pages 1–6, 12 2016.

[97] Amirhossein Gharib, Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. An Evaluation Framework for Intrusion Detection Dataset. *2016 International Conference on Information Science and Security (ICISS)*, pages 1–6, 2016.

[98] Boris Ginzburg and Alex Kesselman. Performance analysis of A-MPDU and A-MSDU aggregation in IEEE 802.11n. In *2007 IEEE Sarnoff Symposium*, pages 1–5, April 2007.

[99] K. Goeschel. Reducing false positives in intrusion detection systems using data-mining techniques utilizing support vector machines, decision trees, and naive bayes for off-line analysis. In *SoutheastCon 2016*, pages 1–6, March 2016.

[100] P. Gogoi, D. K. Bhattacharyya, B. Borah, and J. K. Kalita. A survey of outlier detection methods in network anomaly identification. *The Computer Journal*, 54 (4):570–588, 2011.

[101] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (HBOS): A fast unsupervised anomaly detection algorithm. *KI 2012: Advances in artificial intelligence: 35th Annual German Conference on AI*, pages 59–63, 2012.

[102] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *arXiv:1708.06733 [cs]*, August 2017. arXiv: 1708.06733.

[103] S. Gueron, A. Langley, and Y. Lindell. Aes-gcm-siv: Nonce misuse-resistant authenticated encryption. RFC 8452, RFC Editor, April 2019. URL `http://www.rfc-editor.org/rfc/rfc8452.txt`.

[104] Shay Gueron and Vlad Krasnov. The fragility of AES-GCM authentication algorithm. In *2014 11th International Conference on Information Technology: New Generations*, pages 333–337, April 2014. doi: 10.1109/ITNG.2014.31.

[105] Shay Gueron and Yehuda Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. Technical Report 102, 2015. URL `http://eprint.iacr.org/2015/102`.

[106] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Specification and analysis. Technical Report 168, 2017. URL http://eprint.iacr.org/2017/168.

[107] S. Guha, N. Mishra, et al. Robust random cut forest based anomaly detection on streams. In *Proc. of The 33rd Int. Conf. on Machine Learning*, volume 48 of *Proc. of Machine Learning Research*, pages 2712–2721, New York, USA, 2016. PMLR. URL http://proceedings.mlr.press/v48/guha16.html.

[108] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2712–2721, New York, New York, USA, 20–22 Jun 2016. PMLR.

[109] Aleksey Gurtovoy and David Abrahams. The boost C++ metaprogramming library. page 22, 2002.

[110] Peter Hall, Byeong U Park, Richard J Samworth, et al. Choice of neighbor order in nearest-neighbor classification. *The Annals of Statistics*, 36(5):2135–2152, 2008.

[111] James Douglas Hamilton. *Time series analysis*. Princeton university press, Princeton, NJ, USA, 2020.

[112] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In *Advances in Cryptology – CRYPTO 2008*, pages 144–161, Berlin, Heidelberg, 2008. Springer. ISBN 978-3-540-85174-5. doi: 10.1007/978-3-540-85174-5_9.

[113] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[114] Alexander Hartl, Robert Annessi, and Tanja Zseby. A subliminal channel in EdDSA: Information leakage with high-speed signatures. In *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, MIST '17, pages 67–78, New York, NY, USA, October 2017. Association for Computing Machinery. ISBN 978-1-4503-5177-5. doi: 10.1145/3139923.3139925.

[115] Alexander Hartl, Robert Annessi, and Tanja Zseby. Subliminal channels in high-speed signatures. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 9(1):30–53, 2018.

[116] Alexander Hartl, Maximilian Bachl, Joachim Fabini, and Tanja Zseby. Explainability and adversarial robustness for RNNs. In *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 148–156, New York, NY, USA, 2020. IEEE.

[117] Alexander Hartl, Félix Iglesias, and Tanja Zseby. SDOstream: Low-density models for streaming outlier detection. In *ESANN 2020 proceedings*, pages 661–666, 2020.

[118] Alexander Hartl, Joachim Fabini, Christoph Roschger, Peter Eder-Neuhauser, Marco Petrovic, Roman Tobler, and Tanja Zseby. Subverting counter mode encryption for hidden communication in high-security infrastructures. In *The 16th International Conference on Availability, Reliability and Security*, pages 1–11, 2021.

[119] Alexander Hartl, Joachim Fabini, and Tanja Zseby. Separating flows in encrypted tunnel traffic. In *21st IEEE International Conference on Machine Learning and Applications*, pages 609–616. IEEE, 2022.

[120] Alexander Hartl, Félix Iglesias, and Tanja Zseby. dSalmon: High-Speed Anomaly Detection for Evolving Multivariate Data Streams. In *16th EAI International Conference on Performance Evaluation Methodologies and Tools*. ACM, 2023.

[121] Mohammad J. Hashemi, Greg Cusack, and Eric Keller. Towards Evaluation of NIDSs in Adversarial Setting. In *Big-DAMA '19*, pages 14–21, Orlando, FL, USA, 2019. ACM. ISBN 978-1-4503-6999-2. doi: 10.1145/3359992.3366642.

[122] Douglas M. Hawkins. *Identification of outliers*, volume 11. Chapman and Hall London ; New York, 1980.

[123] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. The multi-user security of GCM, revisited: Tight bounds for nonce randomization. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1429–1440, New York, NY, USA, October 2018. Association for Computing Machinery. ISBN 978-1-4503-5693-0. doi: 10.1145/3243734.3243816.

[124] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[125] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. *Advances in Neural Information Processing Systems*, 29:4080–4088, 2016.

[126] IEEE80211ax. 802.11ax Frame Aggregation Enhancements, December 2020. URL `https://www.extremenetworks.com/extreme-networks-blog/802-11ax-frame-aggregation-enhancements/`. Accessed: 2023-05-07.

[127] F. Iglesias, T. Zseby, et al. Outlier detection based on low density models. In *2018 IEEE Int. Conf. on Data Mining Workshops (ICDMW)*, pages 970–979, 2018. doi: 10.1109/ICDMW.2018.00140.

[128] F. Iglesias, T. Zseby, et al. MDCGen: Multidimensional Dataset Generator for Clustering. *Journal of Classification*, 36:599–618, 2019.

[129] Félix Iglesias and Tanja Zseby. Time-activity footprints in ip traffic. *Comput. Netw.*, 107(P1):64–75, October 2016.

[130] Félix Iglesias and Tanja Zseby. Pattern discovery in internet background radiation. *IEEE Transactions on Big Data*, 5(4):467–480, 2017.

[131] Félix Iglesias, Alexander Hartl, Tanja Zseby, and Arthur Zimek. Are network attacks outliers? a study of space representations and unsupervised algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 159–175. Springer, 2019.

[132] Félix Iglesias and Tanja Zseby. Entropy-Based Characterization of Internet Background Radiation. *Entropy*, 17(1):74–101, January 2015.

[133] Félix Iglesias Vázquez, Tanja Zseby, and Arthur Zimek. Outlier detection based on low density models. In *2018 IEEE International Conference on Data Mining Workshops, ICDM Workshops, Singapore, Singapore, November 17-20, 2018*, pages 970–979, 2018.

[134] K. Igoe and J. Solinas. Aes Galois counter mode for the secure shell transport layer protocol. RFC 5647, RFC Editor, August 2009. URL `http://www.rfc-editor.org/rfc/rfc5647.txt`.

[135] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In *Advances in Cryptology – CRYPTO 2012*, pages 31–49, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-32009-5. doi: 10.1007/978-3-642-32009-5_3.

[136] Ulf Johansson and Patrick Gabrielsson. Are traditional neural networks well-calibrated? In *2019 Int. Joint Conf. on Neural Networks (IJCNN)*, pages 1–8, 2019.

[137] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[138] Antoine Joux. Authentication failures in NIST version of GCM. *Comments submitted to NIST Modes of Operation Process*, page 3, 2006.

[139] Ishan Karunanayake, Nadeem Ahmed, Robert Malaney, Rafiqul Islam, and Sanjay K Jha. De-anonymisation attacks on tor: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2324–2350, 2021.

[140] R. Kaur and M. Singh. A survey on zero-day polymorphic worm detection techniques. *IEEE Communications Surveys Tutorials*, 16(3):1520–1549, Third 2014.

[141] Stephen Kent. IP Authentication Header. RFC 4302, RFC Editor, December 2005. URL http://www.rfc-editor.org/rfc/rfc4302. Num Pages: 34.

[142] Stephen Kent. IP Encapsulating Security Payload (ESP). RFC 4303, RFC Editor, December 2005. URL http://www.rfc-editor.org/rfc/rfc4303. Num Pages: 44.

[143] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019.

[144] Aechan Kim, Mohyun Park, and Dong Hoon Lee. Ai-ids: Application of deep learning to real-time web intrusion detection. *IEEE Access*, 8:70245–70261, 2020.

[145] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. Technical Report 106, 2003. URL https://eprint.iacr.org/2003/106.

[146] M. Kontaki, A. Gounaris, et al. Continuous monitoring of distance-based outliers over data streams. In *IEEE 27th Int. Conf. on Data Engineering*, pages 135–146, 2011. doi: 10.1109/ICDE.2011.5767923.

[147] Bjoern Krollner, Bruce J Vanstone, and Gavin R Finnie. Financial time series forecasting with machine learning techniques: A survey. In *ESANN 2010 proceedings*, 2010.

[148] Yuichi Kumano, Shingo Ata, Nobuyuki Nakamura, Yoshihiro Nakahira, and Ikuo Oka. Towards real-time processing for application identification of encrypted traffic. In *2014 Int. Conf. on Computing, Networking and Communications (ICNC)*, pages 136–140. IEEE, 2014.

[149] Robin Kwant, Tanja Lange, and Kimberley Thissen. Lattice klepto. In *International Conference on Selected Areas in Cryptography*, pages 336–354. Springer, 2017.

[150] Himabindu Lakkaraju and Cynthia Rudin. Learning Cost-Effective and Interpretable Treatment Regimes. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 166–175, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[151] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1939–1947, 2015.

[152] Yiming Li, Yong Jiang, Zhifeng Li, and Shu-Tao Xia. Backdoor learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[153] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38 (11):1857–1874, 2005.

[154] Yeon-sup Lim, Hyun-chul Kim, Jiwoong Jeong, Chong-kwon Kim, Ted "Taekyoung" Kwon, and Yanghee Choi. Internet traffic classification demystified: On the sources of the discriminative power. In *Proc. of the 6th Int. Conf.*, Co-NEXT '10, pages 9:1–9:12, New York, NY, USA, 2010. ACM.

[155] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-based anomaly detection. 6(1): 3:1–39, 2012. doi: 10.1145/2133360.2133363.

[156] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[157] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks. *arXiv:1805.12185 [cs]*, May 2018. arXiv: 1805.12185.

[158] Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[159] Richard G Lyons. *Understanding digital signal processing, 3rd edition*. Pearson, London, UK, 2011.

[160] G. Mahalakshmi, S. Sridevi, and S. Rajaram. A survey on forecasting of time series data. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, pages 1–8, 2016.

[161] E. Manzoor, H. Lamba, et al. xStream: Outlier detection in feature-evolving data streams. In *24th ACM SIGKDD Int. Conf. on Know. Discovery and Data Mining*, 2018.

[162] David McGrew and John Viega. The galois/counter mode of operation (gcm). *Submission to NIST Modes of Operation Process*, page 44, 2004.

[163] David A. McGrew and John Viega. The Security and Performance of the Galois/-Counter Mode of Operation (Full Version). Technical Report 193, 2004. URL `http://eprint.iacr.org/2004/193`.

[164] Fares Meghdouri. Datasets Preprocessing, 2021. URL `https://github.com/CN-TU/Datasets-preprocessing`. GitHub repository.

[165] Fares Meghdouri, Tanja Zseby, and Félix Iglesias. Analysis of Lightweight Feature Vectors for Attack Detection in Network Traffic. *Applied Sciences*, 8(11):2196, November 2018.

[166] Fares Meghdouri, Tanja Zseby, and Felix Iglesias Vazquez. Analysis of lightweight feature vectors for attack detection in network traffic. *Applied Sciences*, 8(11), 2018.

[167] Fares Meghdouri, Félix Iglesias Vázquez, and Tanja Zseby. Shedding light in the tunnel: Counting flows in encrypted network traffic. In *2021 Int. Conf. on Data Mining Workshops (ICDMW)*, pages 798–804. IEEE, 2021.

[168] Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. Revisiting the calibration of modern neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[169] Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2019.

[170] Nour Moustafa and Jill Slay. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *MilCIS*, pages 1–6, November 2015.

[171] Nour Moustafa and Jill Slay. The evaluation of network anomaly detection systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Inf. Sec. J.: A Global Perspective*, 25(1-3):18–31, April 2016. ISSN 1939-3555.

[172] Pramita Sree Muhuri, Prosenjit Chatterjee, Xiaohong Yuan, Kaushik Roy, and Albert Esterline. Using a long short-term memory recurrent neural network (lstm-rnn) to classify network attacks. *Information*, 11(5):243, 2020.

[173] Hoang Vu Nguyen and Vivekanand Gopalkrishnan. Feature extraction for outlier detection in high-dimensional spaces. In Huan Liu, Hiroshi Motoda, Rudy Setiono, and Zheng Zhao, editors, *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, volume 10 of *Proceedings of Machine Learning Research*, pages 66–75, Hyderabad, India, 21 Jun 2010. PMLR.

[174] Hoang Vu Nguyen, Hock Hee Ang, and Vivekanand Gopalkrishnan. Mining outliers with ensemble of heterogeneous detectors on random subspaces. In *International Conference on Database Systems for Advanced Applications*, pages 368–383, Berlin, Heidelberg, 2010. Springer.

[175] Yohei Okada, Shingo Ata, Nobuyuki Nakamura, Yoshihiro Nakahira, and Ikuo Oka. Application identification from encrypted traffic based on characteristic changes by encryption. In *2011 IEEE Int. Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 1–6. IEEE, 2011.

[176] Julian D Olden and Donald A Jackson. Illuminating the “black box”: a randomization approach for understanding variable contributions in artificial neural networks. *Ecological Modelling*, 154(1):135–150, August 2002. ISSN 0304-3800. doi: 10.1016/S0304-3800(02)00064-9.

[177] Julian D Olden, Michael K Joy, and Russell G Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(3):389–397, November 2004. ISSN 0304-3800. doi: 10.1016/j.ecolmodel.2004.03.013.

[178] Adam Paszke, Sam Gross, Soumith Chintala, et al. Automatic differentiation in PyTorch. page 4, 2017.

[179] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[180] M. I. Petrovskiy. Outlier detection algorithms in data mining systems. *Programming and Computer Software*, 29(4):228–237, Jul 2003.

[181] Tomáš Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, February 2016.

[182] D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[183] Huming Qiu, Hua Ma, Zhi Zhang, Alsharif Abuadbba, Wei Kang, Anmin Fu, and Yansong Gao. Towards a critical evaluation of robustness for deep learning backdoor countermeasures. *arXiv preprint arXiv:2204.06273*, 2022.

[184] Michael O Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Inst of Tech Cambridge Lab for Computer Science, 1979.

[185] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. pages 427–438, 2000.

[186] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018. URL `http://www.rfc-editor.org/rfc/rfc8446.txt`.

[187] Shahbaz Rezaei and Xin Liu. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine*, 57(5):76–81, 2019.

[188] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *KDD*, pages 1135–1144, San Francisco, California, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778.

[189] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21 (2):120–126, 1978.

[190] Pieter Robyns, Peter Quax, and Wim Lamotte. Injection attacks on 802.11n MAC frame aggregation. In *Proc. of the 8th ACM Conf. on Security & Privacy in Wireless and Mobile Networks*, pages 1–11, New York, NY, USA, June 2015. ACM. ISBN 978-1-4503-3623-9.

[191] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[192] James Rotton and James Frey. Air pollution, weather, and violent crimes: Concomitant time-series analysis of archival data. *Journal of personality and social psychology*, 49(5):1207, 1985.

[193] Markku-Juhani O. Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. Technical Report 202, 2011. URL `https://eprint.iacr.org/2011/202`.

[194] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):1–21, 2015.

[195] Saad Saleh, Junaid Qadir, and Muhammad U Ilyas. Shedding light on the dark corners of the internet: A survey of tor research. *Journal of Network and Computer Applications*, 114:1–28, 2018.

[196] J. Salowey, A. Choudhury, and D. McGrew. Aes Galois counter mode (gcm) cipher suites for TLS. RFC 5288, RFC Editor, August 2008. URL `http://www.rfc-editor.org/rfc/rfc5288.txt`.

[197] Nicholas I. Sapankevych and Ravi Sankar. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[198] Saket Sathe and Charu C. Aggarwal. Subspace outlier detection in linear time with randomized hashing. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 459–468, 2016.

[199] Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart. Surreptitiously weakening cryptographic systems. Technical Report 097, 2015. URL `http://eprint.iacr.org/2015/097`.

[200] E. Schubert, A. Zimek, and H.-P. Kriegel. Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection. 28(1):190–237, 2014. doi: 10.1007/s10618-012-0300-z.

[201] Erich Schubert and Arthur Zimek. Elki: A large open-source library for data analysis-elki release 0.7. 5" heidelberg". *arXiv preprint arXiv:1902.03616*, 2019.

[202] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[203] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *ICISSP*, pages 108–116, Funchal, Madeira, Portugal, 2018. SCITEPRESS.

[204] Ryan Sheatsley, Nicolas Papernot, Michael J Weisman, Gunjan Verma, and Patrick McDaniel. Adversarial examples for network intrusion detection systems. *Journal of Computer Security*, (Preprint):1–26, 2022.

[205] Reza Shokri et al. Bypassing backdoor detection algorithms in deep learning. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 175–183. IEEE, 2020.

[206] Jocelyn Sietsma. Neural net pruning-why and how. In *Proceedings of the International Conference on Neural Networks*, pages 325–333, San Diego, CA, 1988. IEEE.

[207] Gustavus J Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology*, pages 51–67. Springer, 1984.

[208] Gustavus J. Simmons. The subliminal channel and digital signatures. In *Advances in Cryptology*, pages 364–378. Springer Berlin Heidelberg, April 1984. doi: 10.1007/3-540-39757-4_25.

[209] Gustavus J Simmons. Subliminal communication is easy using the dsa. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 218–232. Springer, 1993.

[210] Gustavus J Simmons. The history of subliminal channels. *IEEE Journal on Selected Areas in Communications*, 16(4):452–462, 1998.

[211] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[212] StackExchange Cross Validated. Feature selection using deep learning?, 2016. URL `https://stats.stackexchange.com/questions/250381/feature-selection-using-deep-learning`.

[213] StackExchange Cross Validated. neural networks - Variable importance in RNN or LSTM, 2019. URL `https://stats.stackexchange.com/questions/191855/variable-importance-in-rnn-or-lstm`.

[214] Guang-Lu Sun, Yibo Xue, Yingfei Dong, Dongsheng Wang, and Chenglong Li. An novel hybrid method for effectively classifying encrypted traffic. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5. IEEE, 2010.

[215] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David Kaeli. Summarizing cpu and gpu design trends with product data. *arXiv e-prints*, 2020.

[216] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[217] Martin Teuffenbach, Ewa Piatkowska, and Paul Smith. Subverting network intrusion detection: Crafting adversarial examples accounting for domain-specific constraints. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 301–320. Springer, 2020.

[218] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *Proc. of the second ACM conf. on Wireless network security*, pages 79–86, New York, NY, USA, March 2009. ACM. ISBN 978-1-60558-460-7.

[219] Mathy Vanhoef and Frank Piessens. Practical verification of WPA-TKIP vulnerabilities. In *Proc. of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS '13, pages 427–436, New York, NY, USA, May 2013. ACM. ISBN 978-1-4503-1767-2.

[220] Mathy Vanhoef and Frank Piessens. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1313–1328, New York, NY, USA, October 2017. ACM. ISBN 978-1-4503-4946-8.

[221] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *Int. Journal of Network Management*, 25(5):355–374, 2015.

[222] J. Viega and D. McGrew. The use of Galois/counter mode (gcm) in IPsec encapsulating security payload (esp). RFC 4106, RFC Editor, June 2005. URL `http://www.rfc-editor.org/rfc/rfc4106.txt`.

[223] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Evaluation of recurrent neural network and its variants for intrusion detection system (ids). *International Journal of Information System Modeling and Design (IJISMD)*, 8(3): 43–63, 2017.

[224] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson,

C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[225] Andrew J Viterbi. A personal history of the viterbi algorithm. *IEEE Signal Processing Magazine*, 23(4):120–142, 2006.

[226] Alina Vlădutu, Dragoş Comăneci, and Ciprian Dobre. Internet traffic classification based on flows' statistical properties with machine learning. *Int. Journal of Network Management*, 27(3):e1929–n/a, 2017.

[227] Gernot Vormayr, Joachim Fabini, and Tanja Zseby. Why are my flows different? a tutorial on flow exporters. *IEEE Communications Surveys & Tutorials*, 22(3): 2064–2103, 2020.

[228] Stephen F Weng, Jenna Reps, Joe Kai, Jonathan M Garibaldi, and Nadeem Qureshi. Can machine-learning improve cardiovascular risk prediction using routine clinical data? *PloS one*, 12(4):1–14, 2017.

[229] Doug Whiting, Russ Housley, and Niels Ferguson. AES encryption & authentication using CTR mode & CBC-MAC. Technical Report IEEE 802.11-02/001r0, January 2002. URL `https://web.cs.ucdavis.edu/~rogaway/ocb/whf02.pdf`.

[230] Nigel Williams, Sebastian Zander, and Grenville Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, October 2006.

[231] Nigel Williams, Sebastian Zander, and Grenville Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, October 2006.

[232] Thomas Wouters. Answer to "what is the maximum recursion depth in python, and how to increase it?", 2010. URL `https://stackoverflow.com/a/3323013`. Stackoverflow discussion.

[233] D. Yang, E. Rundensteiner, et al. Neighbor-based pattern detection for windows over streaming data. In *Proc. of the 12th Int. Conf. on Extending Database Tech.: Advances in Database Tech.*, EDBT'09, pages 529–540, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-422-5. doi: 10.1145/1516360.1516422.

[234] Selim F Yilmaz and Suleyman S Kozat. Pysad: A streaming anomaly detection framework in python. *arXiv preprint arXiv:2009.02572*, 2020.

[235] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, 5: 21954–21961, 2017.

[236] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27*, pages 3320–3328. MIT Press, 2014.

[237] William J Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950.

[238] Adam Young and Moti Yung. The dark side of "black-box" cryptography or: Should we trust capstone? In *Advances in Cryptology — CRYPTO '96*, pages 89–103, Berlin, Heidelberg, 1996. Springer. ISBN 978-3-540-68697-2. doi: 10.1007/3-540-68697-5_8.

[239] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In *Advances in Cryptology — EUROCRYPT '97*, pages 62–74, Berlin, Heidelberg, 1997. Springer. ISBN 978-3-540-69053-5. doi: 10.1007/3-540-69053-0_6.

[240] D. Zhang, K. Lee, and I. Lee. Periodic Pattern Mining for Spatio-Temporal Trajectories: A Survey. In *2015 10th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 306–313, New York, NY, USA, November 2015. IEEE.

[241] Ethan Zhang and Yi Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9.

[242] J. Zhang and M. Zulkernine. Anomaly based network intrusion detection with unsupervised outlier detection. In *2006 IEEE International Conference on Communications*, volume 5, pages 2388–2393, 2006.

[243] Shuang Zhao, Jing Li, Jianmin Wang, Zhao Zhang, Lin Zhu, and Yong Zhang. attackgan: Adversarial attack against black-box ids using generative adversarial networks. *Procedia Computer Science*, 187:128–133, 2021.

[244] Xianfeng Zhao and Ning Li. Reversible watermarking with subliminal channel. In *International Workshop on Information Hiding*, pages 118–131. Springer, 2008.

[245] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[246] A. Zimek and P. Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms. 8(6):e1280, 2018. doi: 10.1002/widm.1280.

[247] Arthur Zimek, Matthew Gaudet, Ricardo J.G.B. Campello, and Jörg Sander. Subsampling for efficient and effective unsupervised outlier detection ensembles.

In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 428–436, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747.