Anomaly Detection Algorithms and Techniques for Network Intrusion Detection Systems

Mikhail Mishin

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 03.06.2020

Supervisor

Prof. Alexander Jung

Advisor

M.Sc. Julen Kahles

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Mikhail Mishin

Title Anomaly Detection Algorithms and Techniques for Network Intrusion Detection Systems

Degree programme Computer, Communication and Information Sciences

Major Computer Science **Code of major** SCI3042

Supervisor Prof. Alexander Jung

Advisor M.Sc. Julen Kahles

Date 03.06.2020 **Number of pages** 79+29 **Language** English

Abstract

In recent years, many deep learning-based models have been proposed for anomaly detection. This thesis presents a comparison of selected deep autoencoding models and classical anomaly detection methods on three modern network intrusion detection datasets. We experiment with different configurations and architectures of the selected models, as well as aggregation techniques for input preprocessing and output postprocessing. We propose a methodology for creating benchmark datasets for the evaluation of the methods in different settings. We provide a statistical comparison of the performance of the selected techniques. We conclude that the deep autoencoding models, in particular AE and VAE, systematically outperform the classic methods. Furthermore, we show that aggregating input network flow data improves the overall performance. In general, the tested techniques are promising regarding their application in network intrusion detection systems. However, secondary techniques must be employed to reduce the high numbers of generated false alarms.

Keywords Anomaly detection, network intrusion detection, neural networks, autoencoders, network flow aggregation, semi-supervised

Acknowledgements

This Master's thesis has been funded by Ericsson Finland and has been carried out as a research project within its R&D division. I am deeply grateful for the provided opportunity which gave me the freedom to pursue a topic of my own interest.

I would like to thank Professor Alex Jung and my advisor Julen Kahles for their thorough guidance throughout this project. Furthermore, I would like to thank my manager Jarno Kyykkä and my colleagues at Ericssion, Juha Törrönen, Tony Hämmäinen, and Juha Eskonen for their assistance and helpful discussions.

Last but not least, I am grateful to my dear family and friends, without whose love and support this thesis may never have been completed.

Espoo, 03.06.2020

Mikhail Mishin

Contents

Abbreviations

AE	Autoencoder
AEGMM	Autoencoding Gaussian Mixture Model
ANIDS	Anomaly-based Network Intrusion Detection System
ANN	Artificial Neural Network
ATK	Attack
C&C	Command and Control
CNN	Convolutional Neural Network
CVAE	Conditional Variational Autoencoder
DDoS	Distributed Denial-of-Service
DL	Deep Learning
DoS	Denial-of-Service
EM	Expectation-Maximization algorithm
FDL	File Download
FTP	File Transfer Protocol
GMM	Gaussian Mixture Model
HPS	Horizontal Port Scan
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IF	Isolation Forest
IoT	Internet of Things
IP	Internet Protocol
IRC	Internet Relay Chat Protocol
ISP	Internet Service Provider
kNN	k-Nearest Neighbor
LSTM	Large Short Term Memory
MAH	Mahalanobis Distance
ML	Machine Learning
MLP	Multilayer Perceptron
MQTT	Message Queuing Telemetry Transport Protocol
NIDS	Network Intrusion Detection System
OHE	One-Hot Encoding
OSI	Open Systems Interconnection
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
SMTP	Simple Mail Transfer Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VAE	Variational Autoencoder

1 Introduction

1.1 Motivation and Context

Telecommunication technologies are evolving rapidly, making the world more connected each year. New standards and technologies, such as 5G - the fifth-generation standard for cellular networks - and Internet-of-Things (IoT) devices, are getting widely adopted in modern industries, including automotive transport applications, smart agricultural farming, and public safety services [1]. The increasing demand for connectivity necessitates raising the standards for security. With more at stake, network security solutions of tomorrow will have to become more automated, scalable, and reliable [2].

There exist many different approaches to network security; for instance, firewalls, email security, and anti-malware software are some of the standard methods for protecting personal computers. In this work, we are interested in the approach called network intrusion detection. The purpose of a network intrusion detection system (NIDS) is to monitor the network's activity and detect in real time various harmful events, such as misconfigurations of network devices or cyberattacks.

A popular approach, also adopted in this work, is to apply anomaly detection techniques to network intrusion detection [3, 4]. An anomaly-based network intrusion detection system (ANIDS) detects malicious network activities by searching for abnormal patterns in network traffic. Figure 1 illustrates the typical flow of an ANIDS. Such systems have many appealing properties, such as detecting novel zero-day attacks. However, certain aspects make a successful deployment of an ANIDS in the real-world environment difficult. According to Sommer et al. [5], the challenges include "a high cost of errors, lack of labeled data, high complexity and variability in input data, and fundamental difficulties for evaluating the system." To overcome these challenges, we need to understand better both the system under protection, as well as the capabilities and limitations of the detection process.

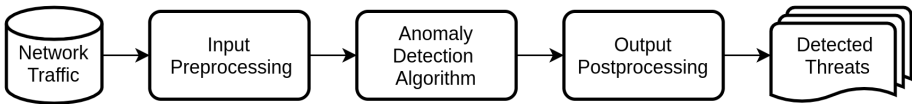


Figure 1: Typical flow of an anomaly-based network intrusion detection system (ANIDS)

Like in any machine learning (ML) application, data is an essential part of the problem. A network intrusion dataset, containing traffic data collected from some network (typically with traces of executed attacks), is necessary for the development and evaluation of anomaly-based network intrusion detection techniques. Due to the complex nature of computer networks, each network intrusion dataset has its own specific characteristics and idiosyncrasies. For example, one dataset may contain traffic from a university network, and another from an industrial IoT network. The statistical nature of the traffic is very different in both cases, as well as the relevant cyberthreats. For this reason, researchers commonly agree that it is crucial to use

several datasets to make any general conclusions in this domain [5, 6]. Fortunately, in the few recent years, a number of representative network intrusion datasets have been created and made publicly available [7, 8, 9, 10]. The availability of modern network intrusion datasets opens new opportunities for research.

At the same time, the research on anomaly detection methods has seen significant advancements. Recently, deep learning (DL) techniques for anomaly detection have become increasingly popular among researchers and practitioners [11]. In particular, autoencoding methods based on artificial neural networks (ANNs) have shown state-of-the-art results on the benchmark anomaly detection datasets [12, 13, 14]. Deep learning methods seem to be particularly appealing for network intrusion detection, as they are able to extract rich representations from input and scale well to large datasets [15, 11].

To the best of our knowledge, comprehensive algorithmic studies in the field of network intrusion detection are few and far between. Many papers proposing new methods use only one dataset for evaluation or rely on outdated datasets, some of which have been shown to be unrealistic, such as KDDCUP'99 [16]. Consequently, we observe the potential for more thorough studies that evaluate state-of-the-art anomaly detection techniques on several representative network intrusion datasets.

1.2 Purpose

Given the context presented in Section 1.1, this work provides a comparison of anomaly detection algorithms and techniques for network intrusion detection. Specifically, we compare deep-autoencoding-based models operating in the semi-supervised mode against classical anomaly detection methods. Furthermore, we analyze the difference in performance of the models trained on the network flow and aggregated-flow features. To the best our knowledge, such a study has not been conducted yet.

Our primary aim is to study the baseline detection performance that can be achieved using the selected techniques. Additionally, we pursue gaining useful insight on the semantics of the detection process.

1.3 Research Questions and Methods

In this thesis, we pursue answering the following questions:

- *How do deep-autoencoding-based models stand in comparison to the methods based on alternative paradigms for the task of detecting network traffic anomalies?* We experiment with three kinds of deep autoencoding models: Autoencoder (AE), Variational Autoencoder (VAE), Autoencoding Gaussian Mixture Model (AEGMM); and two classical models: Isolation Forest (IF) and Mahalanobis distance (MAH). We carry out an empirical comparison of the algorithms over three modern network intrusion detection datasets (CTU-13, CICIDS 2017, and IoT-23), aiming to establish if the deep learning methods bring a significant improvement over the classical approaches. We train and evaluate the models in the semi-supervised mode - a simple, yet realistic setting that uses anomalous data only for tuning of hyperparameters.

- *Does using aggregated statistics of network traffic as input features improve the performance of the models?*

  Network traffic data is typically provided in the format of network flows which can be seen as data points that describe single network connections. A straightforward approach to anomaly-based network intrusion detection is to train models using network flow attributes as input features. Another approach is to aggregate network flows by each device, and compute a new set of statistical features that describe network activity at the device level. We refer to this operation as "network flow aggregation". Our motivation behind studying network flow aggregation stems from the fact that several publications state that it may reveal new patterns beneficial for detecting network anomalies [7, 17].

- *What kind of attacks can be efficiently detected using the above techniques?*

  The selected network intrusion detection datasets contain a diverse range of attack scenarios. We investigate which of them can be detected efficiently using the selected anomaly detection techniques based on network flow data.

This study follows a quantitative research approach to answer the proposed research questions. Thus, we base our conclusions on the results obtained on the experimental data using a thorough comparison methodology.

Figure 2 illustrates the research process of this thesis. We began in the stage of *theory*, where we reviewed the literature on the topic, formulated the research questions and explored available datasets. We continued with the *practice* stage, where we designed our methods and conducted experiments. We had to iterate this process several times as new findings, ideas, and software bugs were observed. A few times, we had to go back to theory, studying additional literature and refining our research questions. Once we obtained our final results, we reached the *presentation* stage, where we made conclusions and reported the results.

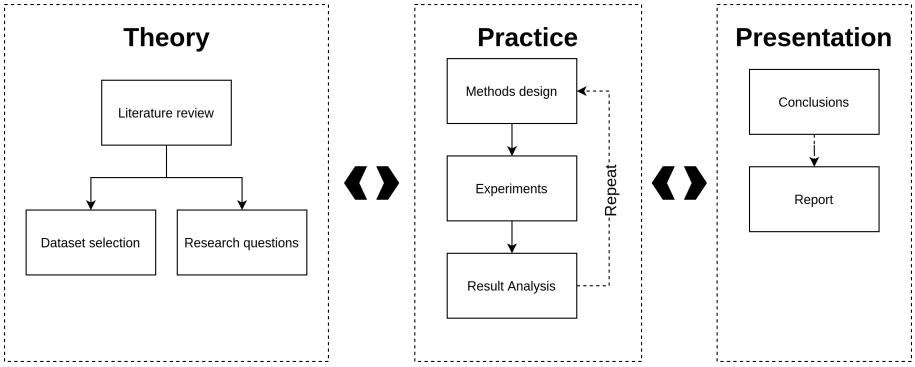


Figure 2: Research process

1.4 Structure of the Thesis

The remainder of this thesis is organized as follows:

- Section 2 reviews the previous work related to the objective of applying anomaly detection methods to network intrusion detection.
- Section 3 provides background information on network security and anomaly detection.
- Section 4 discusses the aspects of anomaly-based network intrusion detection and provides a formalization of the overarching task.
- Section 5 gives an overview of the selected network intrusion datasets, identifying their main properties.
- Section 6 describes our methods and comparison methodology, and gathers the obtained results.
- Section 7 answers the research questions based on the findings and identifies limitations of and future directions for this work.

2 Related Work

Anomaly-based network intrusion detection is a large and actively developing research area. In what follows, we review some of the influential past publications that are related to our work.

Several non-parametric statistical-based methods have been proposed for network intrusion detection. Kruegel and Vigna [18] present an intrusion detection system to detect attacks against web servers and web-based applications. The system takes as input web server logs and derives automatically statistical profiles associated with the referenced web application. The learned profiles are then used to assign an anomaly score for each web request. Mahoney and Chan [19] propose two probabilistic models: the packet header anomaly detection (PHAD) and the application layer anomaly detection (ALAD) model. The models use nonstationary statistical profiling, in which probabilities are estimated based on the time since the last observation of an event rather than its average rate. The PHAD model inspects network packets using the header attributes. The ALAD model inspects server TCP connections using such attributes as the application protocol keywords, TCP flags, IP addresses, and port numbers. The models are evaluated on the 1999 DARPA IDS dataset (cf. Section 5).

Parametric statistical techniques have also been used for modeling network traffic. Simmross-Wattenberg et al. [20] present a two-stage approach for detecting network traffic anomalies. First, the authors use α-stable first-order distributions to model 30-minute windows of network traffic. Second, they apply generalized likelihood ratio test to classify whether the traffic windows are anomalous. The authors focus on detecting two types of attacks, namely floods and flash crowds, and evaluate their method on a custom dataset containing traffic from two university routers. Nguyen et al. [21] propose using the Holt-Winters algorithm, a classic forecasting technique for time series data, to find anomalies in network traffic. The authors apply Holt-Winters to monitor four metrics extracted from the network flow data. Anomalies are detected when any of the metric observations falls outside of the predicted range. For evaluation, the authors create a custom dataset that contains traces of the flooding and port scanning attacks.

Clustering and k-nearest neighbor (kNN) techniques have been proposed for network intrusion detection. Eskin et al. [22] present a framework for unsupervised anomaly detection. First, they apply two mappings to transform the input feature space: a data normalization mapping and a spectrum kernel mapping. Then, they train three algorithms on a set of unlabeled data points from the transformed space. Specifically, they compare a clustering-based model, a kNN model, and a support vector machine (SVM) model. The clustering algorithm produces fixed-width clusters in one pass through the data; the data points in the small clusters are detected as anomalies. For evaluation, the authors use the KDDCUP'99 dataset. García et al. [7] propose a clustering-based method called BClus. They use the Expectation-Maximization algorithm to cluster the aggregated network flow data. In the second stage, they use RIPPER, a propositional rule learning algorithms, to classify the obtained clusters as normal or anomalous. The authors evaluate BClus on the CTU-13 botnet dataset (presented in the same paper).

Several publications have studied the effectiveness of principal component analysis (PCA) for network intrusion detection. Ringberg et al. [23] identify the main challenges of using PCA to detect traffic anomalies. They show that tuning PCA to effectively operate in a real-world environment is difficult and requires more robust approaches than a straightforward one. Brauckhoff et al. [24] show that simple PCA fails to capture the temporal correlation of network traffic and propose replacing it with the Karhunen-Loeve expansion. Kanda et al. [25] present a PCA-based algorithm called ADMIRE for detecting anomalies in flow-based data. ADMIRE uses three-step random projections and an adaptive parameter setting to improve detection performance. The authors use the MAWI dataset for evaluation and conclude that ADMIRE outperforms a classical PCA-based detector.

Numerous classification-based methods for network intrusion detection have been proposed. Moustafa and Slay [26] compare a selection of classification algorithms, including naïve Bayes, decision trees, ANNs, and logistic regression on two datasets: the UNSW-NB15 dataset (presented in the same paper) and KDDCUP'99. Bilge et al. [27] propose a flow-based botnet detection system called Disclosure. The authors train (in a supervised way) a random forest classifier to distinguish command and control (C&C) communication from benign network traffic. To reduce the false positive rate, Disclosure uses external reputation scores, such as Google Safe Browsing. For evaluation, the authors use two network environments: a medium-size university network and a tier 1 internet server provider (ISP) network.

Recently, deep learning-based anomaly detection methods have become popular among researchers [11]. Kwon et al. [28] apply various deep learning models, including multilayer perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) for anomaly detection in network traffic. The authors use three public datasets for evaluation: NSL-KDD, Kyoto Honeypot, and MAWI. Torres et al. [29] propose using a large short term memory (LSTM) network to detect malicious behavior by modeling network traffic as a sequence of states that change over time. The authors evaluate the model on two captures from a university network that contain traces of malicious botnet activities. They show that LSTM achieves good performance overall but struggles with a few cases, where the botnet behavior is not as easily differentiable.

Numerous recent works propose using deep learning-based autoencoding (AE) models for network intrusion detection. Several authors [30, 31, 32] adopt a two-stage approach: first, they use autoencoding-based models to extract compact representations of data points; second, they use the learned representations as input features for standard supervised classifiers, such as logistic regression. A more interesting and practical approach is to train autoencoding models in the fully unsupervised or semi-supervised mode. Mirsky et al. [33] present Kitsune - a novel autoencoder-based NIDS, which can learn to detect network attacks in an online and unsupervised manner. The authors mainly focus on the scalability of the system; however, they claim Kitsune achieves performance comparable to offline anomaly detectors. The evaluation is performed on a custom dataset.

Further, multiple publications [34, 35, 17, 36] propose using more advanced probabilistic versions of autoencoders, such as the variational autoencoder (VAE)

and the conditional variational autoencoder (CVAE). An and Cho [34] propose using the reconstruction probability as the anomaly score for VAE. They show that VAE outperforms AE- and PCA-based methods on the benchmark datasets, including KDDCUP'99. Xu et al. [35] apply VAE for detecting anomalies in seasonal key performance indicators of web applications. Although this security task is different from network intrusion detection, the authors show that their unsupervised method outperforms a supervised ensemble approach by a wide margin.

Despite a surging number of publications proposing novel methods for network intrusion detection, to the best of our knowledge, there seems to be a lack of comprehensive studies, that undertake a rigorous comparison of methods and evaluate them over several representative datasets. A work that gets closer to the topic in hand is that of Falcão et al. [37], which compares a total of 12 unsupervised anomaly detection (statistical, neighbour-based, density-based, and classification-based) algorithms on five network intrusion datasets (KDDCUP'99, NSL-KDD, ISCX2012, ADFA-LD, and UNSW-NB-15). The authors identify the families of algorithms that are more effective for network intrusion detection and the families that are more robust to the choice of configuration parameters. Another study by Skvára et al. [12] compares deep generative autoencoding models (including VAE) against classical algorithms on a large number of benchmark datasets in semi-supervised and unsupervised settings. The authors conclude that the performance of the generative models is determined by the selection process of their hyperparameters. However, the study does not consider the domain of network intrusion detection.

Several previous works [7, 17] propose aggregating network flows; however, to the best our knowledge, no research on the efficacy of this method has been published.

Our work presents a thorough comparison of deep autoencoding-based models and classical methods for network intrusion detection. We train the models in a semi-supervised fashion and evaluate them on three representative modern datasets. Furthermore, we investigate if using the aggregated statistics as input features over the original network flow attributes improves the performance.

3 Background

In this section, we provide a basic introduction to the two separate fields of computer science that are necessary for understanding our work: network security and anomaly detection.

3.1 Network Security

In this subsection, we first explain the basic concepts of networking and principles of communication. We discuss the TCP/IP model and the principles of network communication protocols, such as IP, TCP and UDP. Finally, we look at the different phases of a cyberattack and give an overview of the cyberthreat landscape.

3.1.1 Networking Fundamentals

Communication is a complex process. When a group of people plans to engage in a conversation, they first need to find a place where they can hear each other and agree on the language they will speak. They will likely start with greetings and introductions, following appropriate social norms. The process is similar in computer networks, albeit more complicated and less arbitrary.

A computer network is a group of connected devices that can communicate with each other. Network devices, also called network nodes, can be personal computers, servers, phones, IoT devices, and others. Network nodes are interconnected utilizing some communication media that can be either cable (e.g., twisted pair or optical fiber) or wireless (e.g., WiFi, cellular).

Computer networks are generally classified into [38]:

- *Local Area Networks (LAN)*: A LAN is a small private network, usually limited to a single building.
- *Metropolian Area Network (MAN)*: A MAN is a larger kind of network that typically covers a whole city/metropolitan area.
- *Wide Area Network (WAN)*: A WAN is the largest kind of networks that typically covers whole countries/continents

For historical reasons, there exist two reference models that describe how communication systems operate: the Open Systems Interconnection (OSI) model and the Internet protocol suite, also known as TCP/IP. The former is a more comprehensive reference framework for general networking systems, while the latter is used in the Internet and similar computer networks . The TCP/IP model is sometimes seen as a concise version of the OSI model [39], for which reason we will use it in this thesis.

The TCP/IP model follows the layering principle, dividing a communication system into four abstraction layers, as described in Table 1. Each layer provides some specific functionality to the layer above it and uses the services of the layer below it. For example, the transport layer relies on the internet layer and serves the application layer. Interactions between network devices at the save layer follow

a system of rules called a communication protocol. Communication protocols are designed to exchange specific units of data called protocol data units (PDU). The layer at the destination receives the PDU sent by the layer at the source. For example, the PDU of the internet layer is a data packet.

	Layer	PDU	Function
4	**Application**	Data	Provides software applications with standardized data exchange.
3	**Transport**	Segment, Datagram	Maintains end-to-end communications across the network
2	**Internet**	Packet	Exchanges packets across network boundaries
1	**Link**	Frame	Moves packets between different hosts on the same local network link

Table 1: Layer architecture of the TCP/IP model, adapted from [39]

In what follows, we will describe the main layers and protocols of the TCP/IP model:

- *Link layer.* The link layer is responsible for the physical transmission of data within the local network segment (link) that a device is connected to. The protocols residing in this layer operate this layer operate at the hardware level.

- *Internet layer.* The internet layer is responsible for exchanging data between networks. The principal protocol of this layer is the Internet protocol (IP). IP delivers packets of data from the source node to the destination node based on their IP addresses, which are unique numerical labels that are assigned to all nodes. This process is called routing. There are two versions of the Internet protocol: IPv4 and IPv6. The main difference is that IPv4 uses 32 bit IP addresses, and IPv6 - 128 bit IP addresses. As previously mentioned, data travels across the network in packets due to the physical limit on the amount of data that can be transmitted at one time. An IP packet consists of a header section that contains the source and destination IP addresses, and a data section also called a payload. Figure 3 illustrates the structure of an IPv4 header. IP is a connectionless protocol, meaning that no session information is retained by either the source or the destination node. It is agnostic to the data structures at the transport level.

- *Transport layer.* The transport layer is responsible for end-to-end communication between network nodes. In other words, it ensures that data is transferred from the source node to the destination node, offering control over the reliability of the transmission. The two main protocols residing in the transport layer are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Both protocols segment the data into pieces that get encapsulated in IP packets and sent across the network using the IP protocol.

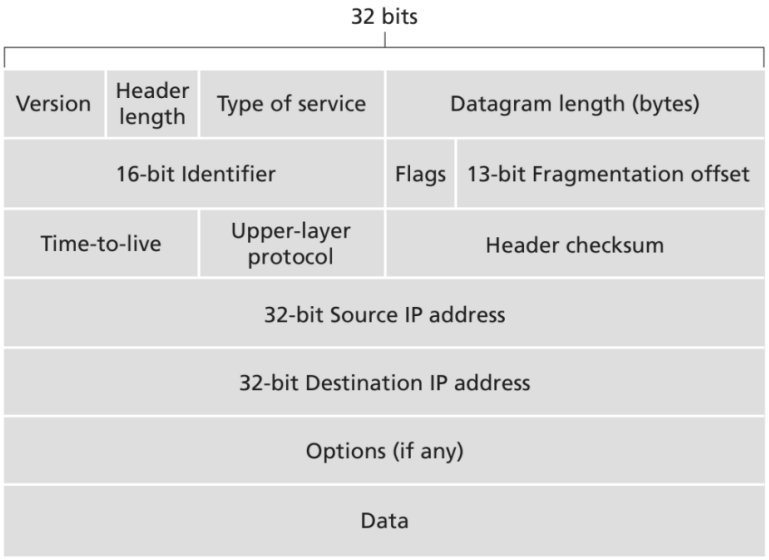


Figure 3: The IPv4 header, taken from *Computer networking : a top-down approach* [38]

TCP provides reliable transmission of data, ensuring that all the packets arrive in order and that there are no duplicated, corrupted, or lost packets. TCP is a connection-oriented protocol, meaning that it establishes a connection between the source node and the destination node before any data is sent and closes it afterward. TCP implements these control mechanisms using special bits in its header called TCP flags; e.g., the ACK flag that is used for acknowledgment. Figure 4 illustrates the structure of a TCP header. TCP is a complex protocol and has high latency due to its reliability features.

UDP is a connectionless transport protocol that provides "unreliable" transmission of data. Unlike TCP, it does not establish a connection between nodes and does not feature error-control mechanisms. UDP is useful in applications like audio or video streaming where low latency is more important than reliability. Figure 5 illustrates the structure of a UDP header.

Transport protocols use network ports to allow multiple conversations between network nodes simultaneously. A network port is a logical construct (represented using an unsigned number) that identifies a specific process or service on a host, which is itself identified by the IP address. Network ports are specified in the transport protocol headers, while IP addresses are specified in the IP headers.

- *Application layer.*

  The application layer provides software applications with data exchange services established by the lower layers. Examples of protocols residing in the application layer are:

  - *Hypertext Transfer Protocol (HTTP)* is extensively used in the World

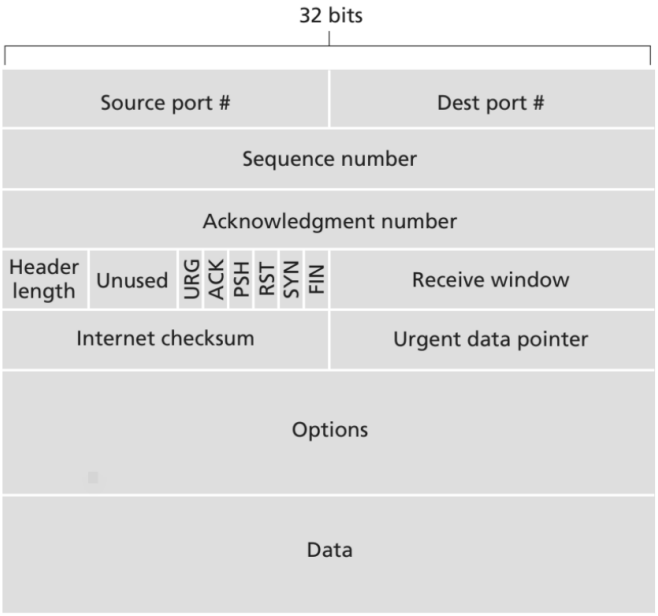


Figure 4: The TCP header, taken from [38]

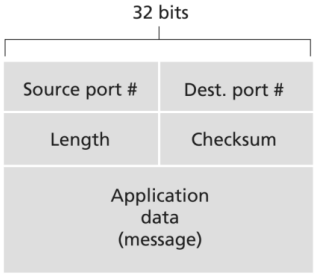


Figure 5: The UDP header, taken from [38]

Wide Web for managing data communication between web clients and servers. HTTPS is a secure version of HTTP.

- *File Transfer Protocol (FTP)* is used for transferring files between a client and server.
- *Simple Mail Transfer Protocol (SMTP)* is used for electronic mail exchange.
- *Internet Relay Chat Protocol (IRC)* is used for communication in the form of text.
- *Message Queuing Telemetry Transport Protocol (MQTT)* is a lightweight messaging protocol for IoT devices, optimized for high latency or unreliable networks.

3.1.2 Cyberthreat Landscape

In this section, we describe common types of cyberattacks and give a brief overview of the cyber threat landscape.

A cyberattack is any kind of malicious activity that targets computer networks or devices [3]. Attackers try to exploit network device or protocol vulnerabilities to gain unauthorized access, steal data, or disrupt the normal functioning of the system. Their motives can vary from personal gain to military intelligence.

A kill chain is a military concept that describes the structure of an attack. Although not all cyber attacks follow every phase of the kill chain, it is a useful tool for attack analysis and defence planning. Table 2 explains the kill chain phases.

#	Phase	Actions
1	**Reconnaissance**	Research, identification and selection of targets
2	**Weaponization**	Pairing remote access malware with exploit into a deliverable payload
3	**Delivery**	Transmission of weapon to target
4	**Exploitation**	Once delivered, the weapon's code is triggered exploiting vulnerable applications or systems
5	**Installation**	The weapon installs a backdoor on a target's system, allowing persistent access
6	**Command & Control**	Outside server communicates with the weapons
7	**Actions on Objective**	The attacker works to achieve the objective of the intrusion

Table 2: Phases of the intrusion kill chain, adapted from [40]

Some common types of cyberattacks include:

- *Network scanning.* In a network scanning attack, the adversary attempts to discover listening ports on the devices in a network. To achieve it, the adversary selects a list of port numbers and tries to establish a connection on each port. More specifically, there are three kinds of port scanning attacks:
  - *Vertical scan:* a scan against a group of IPs for a single port
  - *Horizontal scan:* a scan against a single IP for a group of ports.
  - *Box scan:* a combination of the above.

  The purpose of network scanning is reconnaissance: the adversary's goal is to gather information on the target infrastructure, which will be utilized in the next phase of the attack.

- *Denial-of-Service (DoS) attacks.* In a DoS attack, the adversary attempts to disrupt the normal functionality of a network service, typically to make it unavailable for the users. DoS attacks can target a single computer, e.g., a bank server, or an entire network. DoS attacks are typically executed by flooding the target with traffic, exhausting its capacity to process it. When a DoS attack is carried out by multiple computers, it is called a Distributed

Denial-of-Service attack (DDoS). Naturally, the more resources are used in the attack, the more powerful and disruptive it is.

There are three main categories of DDoS attacks:

- *Application layer attacks.* This category of DDoS attacks target the application layer, where common web activities take place. For example, in an HTTP flood attack, the adversary uses his computational resources to overwhelm the target server with HTTP requests, until it is unable to process normal traffic, as depicted in Figure 6. Application layer attacks are particularly effective because they deplete the server as well as the network resources. Furthermore, they can be hard to detect in time if the malicious requests resemble normal user activity [41].

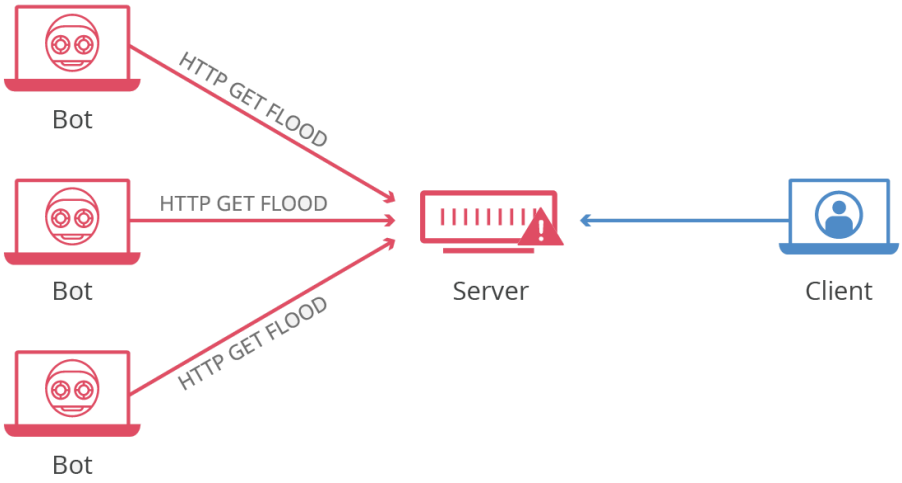


Figure 6: HTTP Flood attack, taken from [41]

- *Protocol layer attacks.* This category of DDoS attacks exploit weaknesses in the transport and internet protocols. For example, in a SYN Flood attack, the adversary misuses the TCP protocol by sending a large number of connection requests to the target at once, never finalizing the connection processes. Other examples of protocol-level DDoS attacks include ACK Flood, SYN-ACK Flood, and UDP flood.
- *Volumetric attacks.* This category of DDoS attacks attempts to consume all the network resources of the target by sending a massive amount of traffic to the target. Often the adversary employs some technique to amplify the amount of data that is sent. For example, in a DNS amplification attack, the attacker sends fake requests to open DNS servers, making them send large replies to the target.

- *Botnets.* A botnet is a group of network devices, each of which has been compromised by an adversary, falling under his control. In this case, each compromised device is called a bot, and the adversary is called the botmaster. The botmaster can use various methods to create bots, like guessing or stealing

authentication credentials or installing a piece of malware. The power of botnets lies in their ability to self-propagate; in other words, the existing bots can "recruit" new bots in their surrounding network. With an efficient recruitment method, the attacker can grow the botnet at an exponential rate. Once large enough, botnets can be used to carry out various malicious activities, such as DDoS attacks, spam distribution, and cryptomining.

To launch an attack, the attacker needs to be able to send commands to the bots. The communication happens through standard network protocols, such as HTTP or IRC. There are two main control structures:

– *Client/server.* This is a centralized control structure where the bots receive commands from the Command and Control (C&C) servers (Figure 7). The advantage of this architecture (from the attacker's point of view) is its simplicity: operating the botnet is straightforward and fast. The disadvantage is that it has a single point of failure - if the defenders manage to identify and bring down the C&C servers, the whole botnet is disrupted.

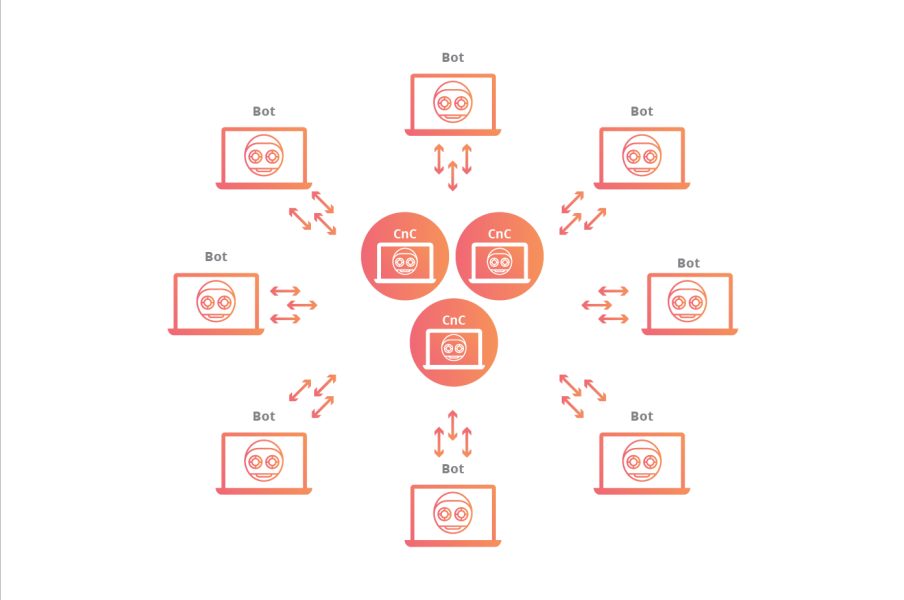


Figure 7: Client/server botnet architecture, taken from [41]

– *Peer-to-peer (P2P).* This is a decentralized control structure - bots take the role of control centers, sending commands to their peers and receiving commands themselves, as illustrated in Figure 8. The advantage of this architecture is that it does not have a single point of failure: bringing down one bot does not disrupt the whole botnet. The price to pay is the increased latency and difficulty of controlling the botnet.

In 2016, one of the largest DDoS attacks in history was carried out using an IoT botnet that was later named "Mirai" [42]. IoT devices often have serious

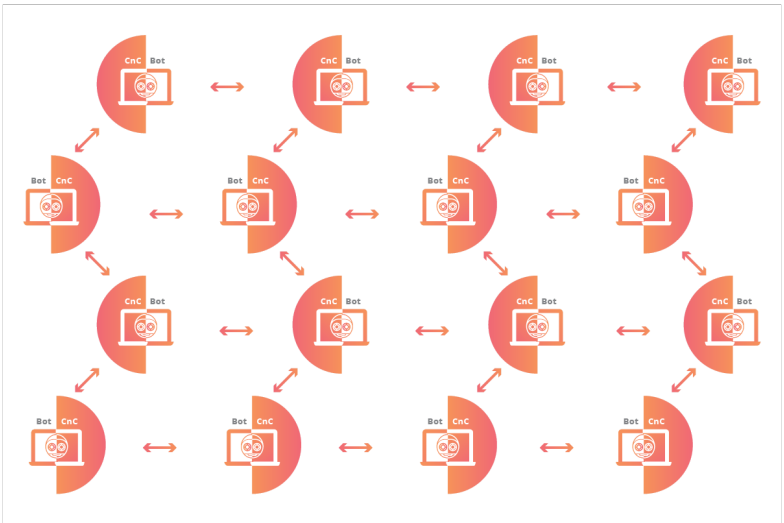


Figure 8: Peer-to-peer botnet architecture, taken from [41]

security vulnerabilities that makes them perfect bot candidates. The Mirai creators used a very simple, yet astonishingly successful technique to recruit new bots; in essence, they were trying to login into devices using a predetermined list of username and password pairs. The Mirai threat has not been stopped; additionally, more sophisticated versions of the botnet, such as "Okiru", have emerged since the first attack. ENISA Threat Landscape Report 2018 [43] ranks IoT botnets among the top modern cyberthreats.

3.2 Anomaly Detection

We now define the problem of anomaly detection and discuss some of its aspects and challenges. We then give an overview of the existing anomaly detection techniques. Finally, we provide a brief theoretical background of the algorithms used in this work.

3.2.1 What is Anomaly Detection

Anomaly detection is a key task of machine learning and data mining. According to Chandola et al. [44], "anomaly detection is the problem of finding observations or patterns in data that do not conform to expected behavior". We refer to these nonconforming patterns as anomalies. In practice, anomaly detection is used to solve detection problems of some kind, usually concerning defects, fraud, or other undesirable events. Example applications include credit card fraud, medical disease diagnosis, natural disaster prediction, and, last but not least, network intrusion detection. The main aspects of an anomaly detection problem are the nature of input data, the type of anomalies, and the availability of labels. All of them have to be taken into account when choosing a suitable anomaly detection technique.

As in a classic machine learning task, data in an anomaly detection problem usually takes the form of a table, image, text, or sequence. Often there is a need to perform feature engineering, which might require domain expert knowledge. The nature of data might limit the choice of a suitable technique; for example, text mining methods are unlikely to be directly applicable to image data. In this thesis, we are dealing with tabular data that has mixed categorical and numeric features. The data points and their attributes are further discussed in Sections 4 and 6.1.2.

Anomalies are typically classified into three categories:

- *Point anomaly.* A point anomaly is a single data point that is found anomalous with respect to the rest of the data. For example, a thief makes an expensive purchase with a stolen card from an otherwise low-expenditure bank account.
- *Contextual anomaly.* A contextual anomaly is a single data point that is found anomalous in a certain context. For instance, a thief uses a stolen card to make a regular transaction at an usual time or geographical location.
- *Collective anomaly.* A collective anomaly is a set of data points that is found anomalous with the rest of the data. For example, a thief carries out a series of otherwise regular transactions on an irregular basis, forming a suspicious pattern.

Techniques for detecting point, contextual, and collective anomalies are different, with the majority of existing methods aiming at point anomaly detection, as this notion of anomaly is the simplest. Techniques for detecting contextual or collective anomalies are highly domain-specific. However, it is sometimes useful to reduce contextual or collective anomalies to point anomalies. Considering the stolen-card examples from above, in the contextual anomaly case, we can add a new "location" feature, so that the data points with unusual locations become point anomalies. In the collective anomaly case, we can transform our data points to represent statistics of transactions over certain periods of time; for example, we can calculate a new "number of ATM withdrawals" feature over a one-week period, so that the data points with high values of "number of ATM withdrawals" become point anomalies. In what follows, we will see a similar reduction for network traffic data.

Obtaining labels for an anomaly detection task is typically hard due to the nature of the problem; indeed, in domains like earthquake detection, anomalies are scarce and limited data is available. Furthermore, abnormal behavior can be highly complex and change over time, in which case collecting and labeling enough data to capture it might not be feasible. On the other hand, data describing normal behavior is usually abundant. Depending on the label availability, anomaly detection algorithms can operate in three modes:

- *Supervised mode.* During training, an algorithm has access to both normal and anomalous data points and their corresponding labels. In the stolen-card example from above, the training data contains both normal and fraudulent transactions with the corresponding labels.

- *Semi-supervised mode.* During training, an algorithm has access to normal data points only. In the above example, the training data contains only normal transactions (only one label is present).

- *Unsupervised mode.* During training, an algorithm experiences data without any labels. In the above example, the training data contains normal and possibly fraudulent transactions (no labels are present).

Unsupervised anomaly detection methods typically require making strong assumptions about the data and perform efficiently only when those assumptions hold [22]. At the other end of the spectrum, supervised techniques are guided by the labels during training, similar to classic predictive models, and do not require making strong assumptions. Semi-supervised techniques are the middle ground between the two and are commonly viewed as being more applicable in practice [44, 12]. Semi-supervised models train on (mostly) normal data and can utilize a small number of anomalies for hyperparameter tuning. We adopt the semi-supervised approach in our experiments. No matter which operational mode is used, a labeled test set is required for evaluating the performance of the method in turn.

Anomaly detection algorithms typically assign an anomaly score to each input instance, which is "a measure of the degree to which that instance is considered an anomaly" [44]. In this work, we follow the convention that "the higher the score, the more likely it is that the instance is an anomaly." To flag all anomalous instances, we need to turn anomaly scores into predictions. A simple way to achieve that is to use a detection threshold. All the instances with scores higher than the threshold get flagged as anomalies. The threshold can be selected by analyzing the top few anomalies during the validation stage.

3.2.2 Overview of Techniques

We now give a brief overview of the existing anomaly detection techniques, describing their underlying assumptions, advantages, and disadvantages. We follow the taxonomy provided in the landmark study by Chandola et al. [44], presented in Figure 9. Lastly, we discuss the aspects of creating an anomaly detection pipeline.

- *Classification-based.* In a classification task, we need to specify which of K categories some input belongs to. In the classical supervised setting, a model is trained on a labeled set of data points that contain instances from each class, ideally in the balanced fashion. In the anomaly detection setting, we would have anomalous classes and normal classes, and flag an input as anomalous if its predicted class belongs to the former. However, as discussed in the previous section, fully-supervised approaches have limited application for anomaly detection, ceding the stage to semi-supervised or unsupervised techniques. In these scenarios, we have only normal-data categories: either a single normal class (one-class-based techniques) or multiple classes (multi-class-based techniques). An input is predicted as anomalous if it doesn't fall within the learned

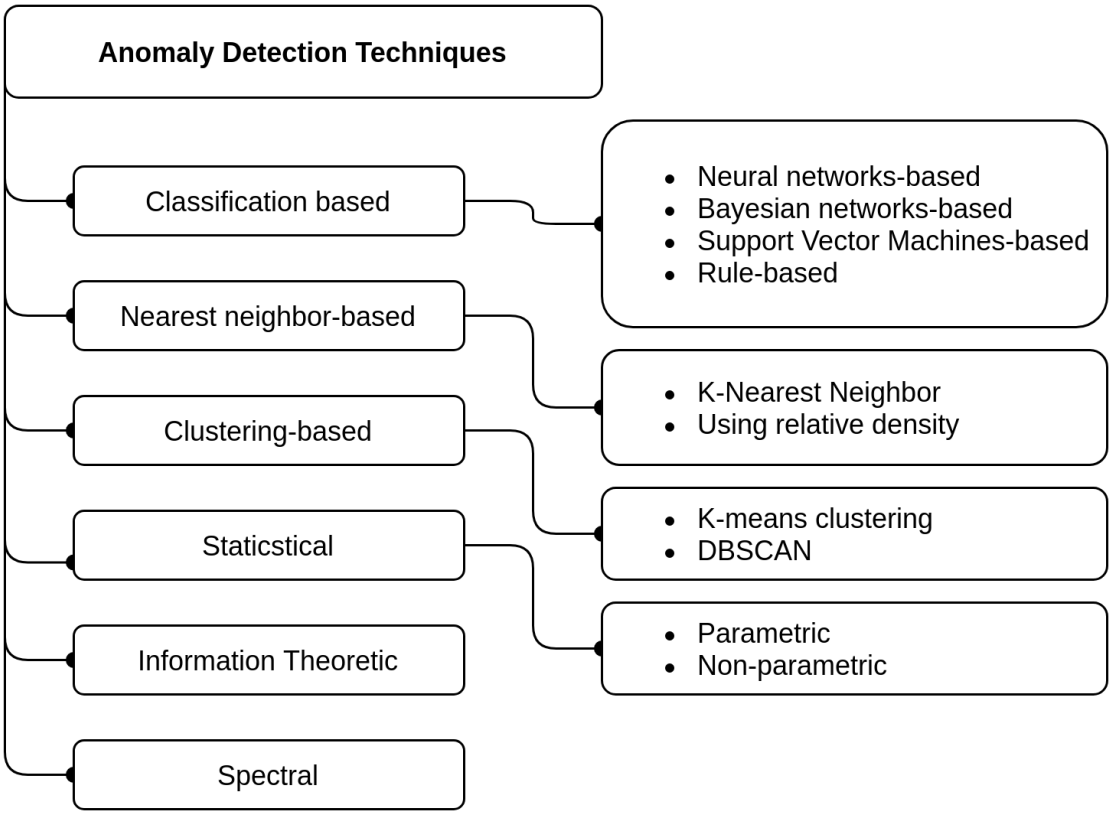
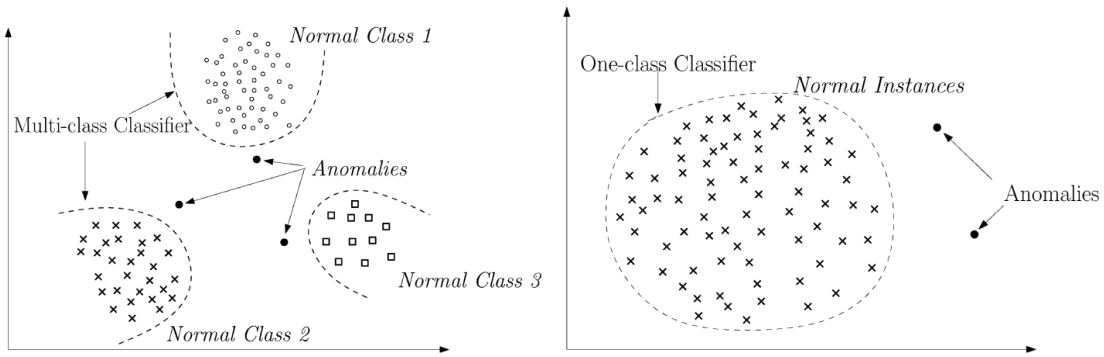


Figure 9: Taxonomy of anomaly detection methods from [44]

decision boundary of the normal classes, as is illustrated in Figure 10. Examples of one-class-based techniques include one-class support vector machines and autoencoders. Multi-class classification anomaly detection models require labels for various normal classes, and, therefore, they cannot operate in the fully-unsupervised mode. Examples of such techniques include artificial neural networks (ANNs), Bayesian networks, and rule-based models, e.g., RIPPER [45].



(a) Multi-class Anomaly Detection (b) One-class Anomaly Detection

Figure 10: Anomaly detection as a classification problem, adapted from [44]

- *Nearest-neighbor-based.* Nearest-neighbor-based anomaly detection techniques analyze an input instance with respect to its local neighborhood. The underlying assumption is that "normal instances occur in dense neighborhoods, while anomalies occur far from their closest neighbours or in sparse neighborhoods" [44]. In the basic variation, the anomaly score of a test instance is defined as the distance to its k^{th} neighbor from the training set. The distance between two data points is calculated using a predefined distance metric. The advantages of nearest-neighbor-based techniques are that they can operate in the unsupervised mode and are fairly robust [46, 12]. One disadvantage is that defining a metric may not be straightforward for complex data types, such as graphs or tabular data with mixed features, which limits the applicability of these techniques. Another disadvantage is that nearest-neighbor-based techniques do not scale well to large datasets, as finding k neighbors of some input among many candidates is computationally expensive. Various approaches to speed up the search exist, such as k-d trees [47], but they alleviate the problem only to a certain extent.

- *Clustering-based.* In a clustering task, we need to group similar inputs into clusters. We can use clustering techniques for anomaly detection if we assume that normal instances form large clusters, while anomalous instances do not belong to any cluster or form small clusters [44]. Examples of clustering algorithms include k-means and DBSCAN [48]. Like nearest-neighbor-based techniques, techniques from this category can operate in the unsupervised mode. Furthermore, clustering-based techniques are fast at the test stage. The disadvantage is that we make strong assumptions about the structure of the data. For example, in case anomalous instances form large clusters, clustering-based techniques will not be effective.

- *Statistical.* Statistical anomaly detection techniques model normal data instances using a stochastic model. The key assumption is that "normal data instances occur in the high probability regions of the model, while anomalies occur in the low probability regions" [44]. Statistical techniques are generally classified into two types: parametric and non-parametric. Parametric techniques use a parametric distribution, e.g,. the Gaussian distribution, to model the data. Non-parametric techniques infer the model structure from the data; for instance, histograms can be used to model the profile of the normal data. Statistical anomaly detection techniques can operate in the unsupervised mode, and are typically robust and provide interpretable results. A disadvantage is that using statistical models requires making strong assumptions about the underlying data distribution. Statistical models can also fail to capture complex interactions between features of multivariate data.

- *Spectral.* Spectral techniques are used to embed data into a lower dimensional subspace. They can be suited for anomaly detection assuming that normal instances and anomalies are separable in the embedding space [44]. The techniques from this category include Principal Component Analysis (PCA) and

T-distributed Stochastic Neighbor Embedding (tSNE). Spectral techniques are not anomaly detection methods per se, but if the anomalies are well-separable in the reduced dimension, the problem can be easily solved. The advantage of spectral techniques is that they can operate in the unsupervised mode. The disadvantage is that they typically have high computational complexity.

When choosing a suitable method for an anomaly detection problem, one should take into account many aspects, such as the underlying assumptions, the availability and accuracy of labels, and the requirements on scalability, speed, and output interpretability. While each problem has its specifics, there are a couple of general considerations.

Firstly, it is a common practice in machine learning to conduct an empirical comparison of several different methods, as identifying the best method based on theory alone is often not possible. Before experimenting with more complex methods, one should always establish a baseline that shows what can be achieved using a simple approach.

Another common practice is to combine several techniques into a single pipeline. For example, one can use PCA to perform dimensionality reduction, cluster the data points using k-means, and train a classification model on the clusters. Once again, comparing such a complex pipeline to a simpler baseline is necessary to measure the improvement (or deterioration) in performance.

3.2.3 Algorithms

We now provide a brief theoretical background on the autoencoding and classical methods for anomaly detection used in this work. We begin by explaining the basic concept of autoencoding. Then, we provide a description of each algorithm, specifying the loss and anomaly score functions. We have adopted some of the notation in this section from the one presented in [12].

According to the definition provided by Chollet [49], "autoencoding is a data compression algorithm where the compression function (encoder) and decompression function (decoder) are learned automatically from data examples rather than engineered by a human". In other words, autoencoders convert the input data into a low-dimensional representation, and reconstruct the original input from it. This approach of learning useful representations from data is known as representation learning. Figure 11 illustrates the basic principle of an autoencoder.

To measure the amount of information loss that occurred during encoding and decoding, we define a loss function between the input and reconstructed instance. Autoencoders are data-specific models, meaning they perform well only on data instances similar to those they were trained on. While this property makes them less useful for real encoding tasks (a music compression algorithm that only works on disco songs from 1970s is hardly practical), it allows us to adopt autoencoding models for anomaly detection. In that case, we optimize the models to reconstruct only normal data instances. As anomalies have not been experienced by the model, their reconstruction loss will be higher than that of normal instances (ideally). The reconstruction loss can be used directly as the anomaly score, establishing a decision

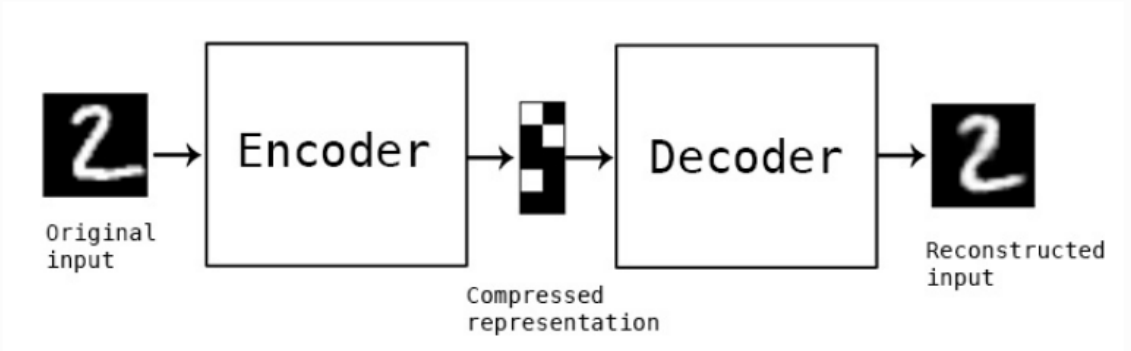


Figure 11: A basic autoencoder, taken from [49]

boundary between the normal and anomaly classes. In this way, autoencoders can be viewed as one-class classification anomaly detection techniques.

Let us describe the deep autoencoding models we used in our experiments. For each model, we identify the objective loss function that is used for optimization and the anomaly score function that measures the degree of deviation from normal.

- *Autoencoder (AE)* models the encoding and decoding functions with two multi layer perceptrons (MLPs). Let $\mathcal{X}$ be the input data space and $\mathcal{Z}$ be the latent space of compressed representations. Then, the encoder represents the mapping $e(\,\cdot\,;\theta_e) : \mathcal{X} \to \mathcal{Z}$ and the decoder represents the mapping $d(\,\cdot\,;\theta_d) : \mathcal{Z} \to \mathcal{X}$, where θ_e and θ_d are the parameters of the AE. Given an input instance x, its reconstruction x' is given by $x' = d(e(x))$. The loss function of AE is the mean squared error (MSE):
$$\mathcal{L}(x, \theta_e, \theta_d) = \|x - x'\|^2 \tag{1}$$
The anomaly score is also given by the MSE:
$$f_{AE}(x) = \mathcal{L}(x, \bar{\theta_e}, \bar{\theta_d}) \tag{2}$$
where $\bar{\theta_e}$, $\bar{\theta_d}$ are the learned parameters of the AE.

- *Variational Autoencoder (VAE)* has a similar architecture to a basic AE, but a completely different mathematical basis [14, 50]. A VAE is a generative model: it learns a function to approximate the underlying data distribution. The encoder network of a VAE maps the input data point x into a latent variable z, which models the parameters (mean and variance) of a latent normal distribution that is assumed to generate x. To reconstruct x, we sample points from that latent distribution and use the decoder to map the points back to the input data space X. More formally, the encoder models a conditional probability $q(z|x;\theta_e)$ and the decoder models a conditional probability $p(x|z;\theta_d)$. Both distributions are assumed to be Gaussian, as well as the prior distribution $p(z)$ of the latent variable, $p(z) \sim \mathcal{N}(0, I)$. The loss function of a VAE has a specific name - the ELBO loss:
$$\mathcal{L}(x, \theta_e, \theta_d) = \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] + \lambda D_{KL}\left(q(z|x)||p(z)\right) \tag{3}$$

The ELBO loss has two components: the reconstruction loss similar to AE, and the KL divergence between the learned latent distribution and the prior distribution, which acts as a regularizer.

The anomaly score can either be measured as the MSE or as the reconstruction probability (the probability that both the input and the reconstructed output are generated by the same process). In the latter case, the anomaly score is given by:

$$f_{VAE}(x) = \mathbb{E}_{q(z|x)} \left[\log p(x|z) \right] \tag{4}$$

- *Auto-Encoding Gaussian Mixture Model (AEGMM)* is a novel state-of-the-art autoencoding model proposed in [13].

  A GMM is a parametric statistical model comprised of K Gaussian components that each has density $p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$, where π_k is a mixing coefficient. Mixing coefficients define the weight of each component and normalize the total probability distribution to 1, $\sum_k \pi_k = 1$. The Expectation-Maximization (EM) algorithm is typically used to fit a GMM.

  An AEGMM performs a dimensionality reduction of the input data using an autoencoder and fits a GMM over the learned low-dimensional space. Figure 12 illustrates the architecture of an AEGMM.

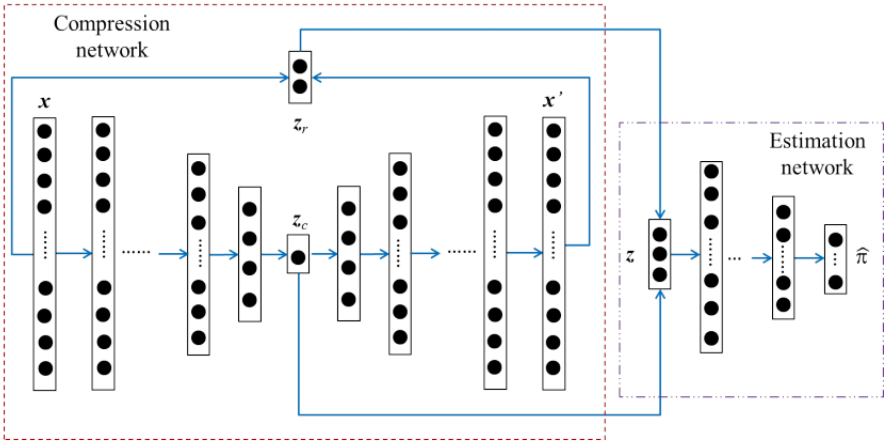


Figure 12: Auto-Encoding Gaussian Mixture Model, taken from [13]

A low-dimensional representation z of an input instance x is obtained by concatenation $z = [z_c, z_r]$, where $z_c = e(x; \theta_e)$ is the compressed representation produced by the encoder, and $z_r = f(x, x')$ is the value of the reconstruction error given by some distance metric $f(\cdot)$. It is possible to use multiple distance functions, e.g., relative Euclidean distance and cosine similarity, in which case z_r is multidimensional. The AEGMM utilizes the estimation network, another MLP parametrized by θ_m, to make mixture-component membership prediction $\hat{\gamma}$ from z (instead of using EM). Given a batch of N samples and their membership predictions, $\forall 1 \leq k \leq K$, we can estimate the parameters in

GMM as follows:

$$\hat{\pi_k} = \sum_{i=1}^{N} \frac{\hat{\gamma_{ik}}}{N}, \quad \hat{\mu_k} = \frac{\sum_{i=1}^{N} \hat{\gamma_{ik}} z_i}{\sum_{i=1}^{N} \hat{\gamma_{ik}}}, \quad \hat{\Sigma_k} = \frac{\sum_{i=1}^{N} \hat{\gamma_{ik}} (z_i - \hat{\mu_k})(z_i - \hat{\mu_k})^T}{\sum_{i=1}^{N} \hat{\gamma_{ik}}} \tag{5}$$

where $\hat{\gamma_i}$ is the membership prediction for the low-dimensional representation z_i, and $\hat{\pi_k}$, $\hat{\mu_k}$, and $\hat{\Sigma_k}$ are the estimated mixture coefficient, mean, covariance for component k in GMM, respectively.

The objective loss function of the AEGMM is given by:

$$\mathcal{L}(x, \theta_e, \theta_d, \theta_m) = \|x - x'\|^2 + \lambda_1 E(z) + \lambda_2 P(\hat{\Sigma}) \tag{6}$$

This loss function includes three components:

- $\|x - x'\|^2$ is L_2 norm that keeps the reconstruction error low
- $E(z)$ is the "energy" of the sample x; in other words, the probability that we could observe x. The equation for the sample energy is given by:

$$E(z) = -\log \left(\sum_{k=1}^{K} \hat{\pi_k} \frac{\exp\left(-\frac{1}{2} (z - \hat{\mu_k})^T \hat{\Sigma_k}^{-1} (z - \hat{\mu_k}) \right)}{\sqrt{|2\pi \hat{\Sigma_k}|}} \right) \tag{7}$$

- $P(\hat{\Sigma})$ is a term penalizing small values of the diagonal entries in the covariance matrix. It is used to avoid degenerate solutions.
- λ_1 and λ_2 are hyperparameters. In practice, these values will be set to $\lambda_1 = 0.1$ and $\lambda_2 = 0.005$.

The anomaly score of the AEGMM is given by the GMM energy sample:

$$f_{AEGMM}(x) = E(z) \tag{8}$$

For further details, refer to the original paper [13].

Let us now describe the classical anomaly detection algorithms:

- *Isolation forest (IF)* [51] is a tree-based model adopted for anomaly detection. An IF builds an ensemble of random trees for a given data sample. Each tree partitions the data points by randomly selecting a feature and then randomly selecting a split value within the range of values for the selected feature. The number of such splits required to isolate an instance is equivalent to the path length from the root node of the tree to the terminating node. The key assumption is that anomalies are easier to isolate than normal instances (cf. Figure 13), and consequently, they have shorter path lengths on average.

  The anomaly score of the AF is given by:

$$f_{IF}(x) = 2^{-E[h(x)]/c(N)} \tag{9}$$

  where $E[h(x)]$ is the path length averaged over all trees and $c(N)$ is a normalizing term.

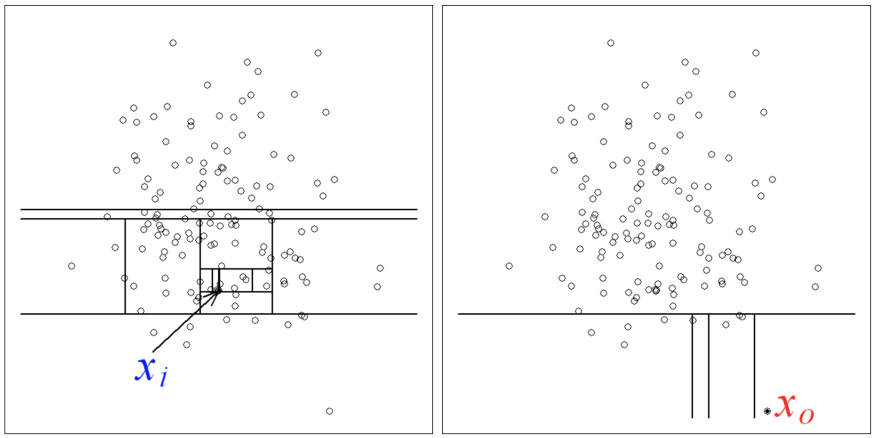


Figure 13: Isolating a normal x_i vs. an anomalous x_o instance (abstraction), taken from [51]

- *Mahalanobis distance (MAH)* [52] calculates an anomaly score which is a measure of distance from the center of the distribution of the features D to a sample x. The distance is given by:

$$f_{MAH}(x) = \sqrt{(x - \hat{\mu})^T \hat{\Sigma}^{-1} (x - \hat{\mu})} \tag{10}$$

  where $\hat{\mu}$ and $\hat{\Sigma}$ are the estimated mean and covariance of the feature distribution, respectively.

  In this work, we use an online implementation of MAH, which means that it starts without knowledge about the distribution of the features and learns as requests arrive [53]. Furthermore, we apply PCA to the original feature distribution to perform dimensionality reduction first. These algorithm-design choices allow us to efficiently apply MAH to large datasets, such as the ones used in our experiments.

Table 3 summarizes the information about the loss and anomaly score functions for each algorithm. We note that all of the selected algorithms are good candidates for anomaly-based network intrusion detection, as they can operate in the unsupervised (or semi-supervised) mode, they are suitable for medium dimensional tabular data, and they are computationally efficient enough to be used in a real-time NIDS.

Algorithm	**Loss function** $\mathcal{L}(x, \theta, \phi)$	**Anomaly score** $f(x)$
AE	MSE	MSE
VAE	ELBO	MSE
AEGMM	AEGMM loss	GMM sample energy
IF	N/A	Avg. path length
MAH	N/A	Mahalanobis distance

Table 3: Algorithms, loss functions, and anomaly scores

4 Network Intrusion Detection

This section describes different types of network intrusion detection systems (NIDS) and their aspects. We present the two standard formats for capturing network traffic data: packet-based and flow-based, and explain why we choose to work with the latter. We discuss the difficulties of representing the IP address information in a way that can be processed by the models. Then, we define network flow aggregation and show how it can be seen as a reduction from collective anomalies to point anomalies. Next, we identify and describe the properties of network intrusion datasets. Finally, we provide a "roadmap" to solving a network intrusion detection problem using anomaly detection techniques, where we give general guidelines on selecting a suitable anomaly detection technique and finding appropriate data.

4.1 Network Intrusion Detection Systems

The task of a NIDS is to detect malicious activities, such as cyberattacks, performed against a network by monitoring its traffic. Based on the style of detection, NIDS are commonly classified into signature-based and anomaly-based [54]:

- A *signature-based NIDS*, also called misuse-based, attempts to detect attacks by searching for their specific patterns in network traffic. Attack patterns or signatures are created and stored in the system beforehand. By design, a signature-based NIDS is robust at detecting known attacks and produces interpretable results. The downside is that it is not able to detect novel attacks with unknown signatures [3]. Another disadvantage is that such a system requires constant signature updates, as new attacks appear and existing ones evolve.

- An *anomaly-based NIDS (ANIDS)* attempts to detect attacks by searching for unusual patterns in network traffic. The ANIDS employs some anomaly detection technique to learn a profile of the benign network traffic data. Any significant deviations from the learned benign profile are flagged as potential malicious activities.

  The main assumption of this approach is that the attacks launched against the network will leave a trace in the network traffic that does not conform to the network's normal behavior. The ANIDS should ideally be able to detect known as well as unknown attacks without any prior knowledge because it does not rely on signatures [3] . However, some challenges arise in practice:

  - Some network devices, such as servers or personal computers, exhibit a very complex and variant behavior. In such environments, it is unrealistic to expect anomaly detection techniques to perform efficiently. If the traffic is highly non-homogeneous, the ANIDS will produce a large number of false alarms (report benign traffic as suspicious), which may limit its usability [5].

- Network traffic may have periodic patterns or drift over time. For example, the amount of traffic may peak during the daytime and fall during the nighttime.
- Specific malicious activities, like C&C communication, can appear to be very similar to the benign traffic, often not without the attacker's efforts.

Table 4 summarises the advantages and disadvantages of signature-based and anomaly-based NIDS. A successful strategy for deploying a NIDS in a real-world environment may include both anomaly-based and signature-based approaches.

Property	**NIDS**	
	Signature	**Anomaly**
Interpretability	+	-
Robustness	+	-
Maintenance	-	+
Unknown attack detection	-	+

Table 4: Advantages and disadvantages of using signature- and anomaly-based NIDS

As this work focuses on anomaly detection techniques, let us discuss a few more points on ANIDS.

It should be mentioned that anomalous patterns in network traffic may not necessarily be caused by attacks and exploits. A misconfiguration of a network device might make its behavior appear suspicious. Detecting such cases is beneficial on its own, as they may be security vulnerabilities.

Another benefit of anomaly detection techniques is that they can be used as tools for creating signature-based detection rules. One way is to try to extract patterns from the detected anomalies, e.g., important features and their corresponding value ranges, which can be used to create signatures.

Arguably, one of the most promising application areas of ANIDS is monitoring IoT networks. Many IoT devices have a relatively simple network profile [55], which can be learned efficiently using anomaly detection techniques. Therefore, we can expect the ANIDS to be more precise and produce less false alarms in IoT network environments.

4.2 Network Traffic Data

There exist two standard formats for capturing network traffic data:

- In the *packet-based* format, a data point is an IP packet, including its payload and header. Thus, it provides a complete picture of the information transferred in the network. It is a common belief that examining payload data may be necessary for detecting certain kinds of attacks. However, one issue of working with packet data is its sheer volume: it may be too resourcefully expensive

to store and process data produced by a large network. Another problem is privacy, as payload data often contains confidential information.

- In the *flow-based* format, In the flow-based format, a data point, called a network flow, only contains meta-information about a connection between network nodes.

  Network flows have to be generated from the packet data using a special networking software tool called a flow collector. The flow collector aggregates IP packets belonging to a single connection and summarizes the information about the sequence of packets into a network flow record. For connectionless protocols, like UDP, a timeout value is used to determine the flow duration. The following five-tuple identifies a network flow: the source IP address, the destination IP address, the source port, the destination port, and the transport protocol [56].

  It should be mentioned that there are two types of network flows: unidirectional and bidirectional. In the unidirectional case, the packets from a network node A to a network node B are aggregated into one flow record, and the response packets from B to A are aggregated into another flow record. In the bidirectional case, all the packets are aggregated into a single flow record.

  Table 5 shows common attributes of a network flow and Figure 14 illustrates a typical output of a flow exporter. Depending on the configuration of the flow exporter, additional attributes may be added to flows, such as bytes per second, mean inter-arrival time of packets, TCP flags, and so on. As we can see, flow-based data typically can be represented as tabular data with mixed features.

#	**Attribute**
1	Timestamp
2	Duration
3	Source IP
4	Destination IP
5	Source port
6	Destination port
7	Protocol
8	Number of bytes transferred
9	Number of packets transferred

Table 5: Common attributes of a network flow.

The advantages and disadvantages of working with packet- and flow-based network data is summarized in Table 6. In the recent years, flow-based techniques have become prevalent in research and industry [57]. Arguably the most important reason is that it is impracticable to collect and process packet data produced by a large network.

```
Date flow start          Duration Proto   Src IP Addr:Port      Dst IP Addr:Port     Packets    Bytes Flows
2010-09-01 00:00:00.459     0.000 UDP     127.0.0.1:24920   ->  192.168.0.1:22126       1       46     1
2010-09-01 00:00:00.363     0.000 UDP     192.168.0.1:22126 ->  127.0.0.1:24920         1       80     1
```

Figure 14: Example output of a flow exporter.

Furthermore, it is easier to work with tabular data. For these reasons, we use network traffic data in the bidirectional flow format in our experiments.

Property	**Format**	
	Packet	**Flow**
Data completeness	+	-
Confidentiality	-	+
Storage costs	-	+
Processing costs	-	+

Table 6: Advantages and disadvantages of packet- and flow-based network data

4.3 Processing of IP Addresses

As explained above, IP addresses are used to (partially) identify network flows. It is important to understand that IP addresses belong to a very large space (2^{32} possible IPv4 addresses, 2^{128} IPv6 addresses), and they lack natural ordering. For these reasons, they cannot be represented using simple encodings, such as one-hot encoding. So, how can machine learning models process them?

Ring et al. [58] identify three approaches for processing IP addresses:

1. *Ignoring IP addresses.* This approach discards the information about IP addresses altogether. In our experiments, we use this method for training the models on the network flow features.

2. *Extracting meaningful features from IP addresses.* This approach does not use IP address information directly, but instead calculates new features from it. In our experiments, we perform aggregation of network flows, which is an example of a technique from this category.

3. *Defining metrics on IP addresses.* This approach defines a distance metric on IP addresses or a mapping to an embedding space. In their work, Ring et al. [58] propose learning IP address embeddings using text mining techniques. We do not use any techniques from this category in our experiments.

4.4 Why Aggregate Network Flows?

The concept of computing aggregated features of network flows is similar to the idea of extracting network flows themselves. Network flows contain aggregated information

about data packets, e.g., “total number of bytes”; thus, network flows are data points that describe network activity at the connection level. In contrast, aggregations of flows describe network activity at the node level.

Aggregated network flow features are extracted in the following way. First, network flows are collected by unique source IP address and during a sliding window, which we refer to as the aggregation window. For example, if we choose a five-minute window, one obtained group can be all the flows sent from IP 172.16.254.1 between 23:25:00 and 23:30:00. The second step is to summarize each grouped sequence of network flows using a new set of descriptive statistics, which we also refer to as aggregated features. We finally obtain data points that correspond to the traffic statistics of a network node within a single aggregation window.

Table 7 presents some examples of the aggregated features that we can calculate. One benefit we obtain when performing this is that the computed statistics are typically numeric. Another benefit is that we retain some of the information about IP addresses and ports by extracting meaningful features from them, such as “number of unique destination IPs”. Our method of aggregating network flows is presented in Section 6.1.2.

#	**Attribute**
1	Timestamp
2	Source IP
3	Number of unique destination IPs
4	Entropy of destination IPs
5	Mean number of packets in a flow
6	Max amount of bytes transferred in a flow

Table 7: Common attributes of an aggregation of network flows on the device IP level

Aggregation of network flows allows us to analyze the problem from a new perspective. It is a common belief that computing aggregated flow statistics may reveal new patterns beneficial for detecting network anomalies [7, 6, 17]; in fact, it can be seen as a reduction from collective anomalies to point anomalies. For instance, in an attack like DDoS, a single bot may initiate a large number of connections in a short burst of time. While the individual network flows may not deviate significantly from the benign profile, the aggregated statistics will, e.g. the total number of flows will be much higher than usual. Albeit its simplicity, a similar argument can be applied to more complicated scenarios, such as C&C communication [7].

An additional benefit of aggregating network flows is that we further reduce the amount of data to process, which can be of a great magnitude even in the case of storing flow-based data. On the other hand, by computing descriptive statistics, we are losing some of the information about individual network flows, which might be helpful for the detection of certain types of attacks.

4.5 Network Intrusion Dataset Properties

Selecting appropriate network intrusion datasets is crucial for the design and evaluation of anomaly detection techniques [5]. During this process, the following properties of network intrusion datasets should be taken into account [6, 3]:

- *Nature of data.* These properties describe the format of the provided network traffic data. As discussed above, network traffic data can be available in the packet- or flow-based format. Depending on the flow collector used, flow-based data can contain different sets of features. Lastly, some attributes, like IP addresses and payloads, may be anonymized or even removed due to privacy reasons.

- *Network environment.* These properties describe the network environment and how the data was collected. Data can be captured in a real network, or it can be emulated using special software. Furthermore, the properties of the captured data will depend on the network type. Examples of different network environments include university networks, internet service provider networks, industrial IoT networks, and so on.

- *Attack types.* These properties describe the present attacks and how they were executed. Ideally, the dataset should contain a diverse and representative set of attacks. The way the attack traces were obtained is also important. For example, attacks can be simulated using a script, or they can be executed by researchers in a lab which arguably produces more realistic data.

- *Label availability.* These properties describe the provided labels and how they were obtained. Accurately labeled data is crucial for evaluation. The ways of obtaining labels include conducting a manual inspection or using a signature-based NIDS. The labeling process is often error-prone, and it is beneficial to have an estimate of the labeling accuracy. Another important aspect is the level of detail provided in the labels. In the basic case, the labels may only indicate malicious and benign data points. More detailed labels may provide specific information on performed malicious activities.

- *Data Volume.* These properties describe the volume and duration of data. The volume is either specified as the total number of the data points (packets, flows) or the physical size in GB. In general, packet-based data has a much larger size compared to flow-based data. Duration refers to the period during which the network traffic data was collected. For studying periodical effects, the duration has to span a sufficient period (e.g., at least a month for comparing weekday vs. weekend patterns) [10].

What are the properties of a perfect network intrusion dataset? According to Ring et al. [6], it has to be "up-to-date, correctly labeled, and publicly available; and has to contain real network traffic with all kinds of attacks and normal user behavior and to span a long period of time." Unfortunately, these requirements are almost

impossible to satisfy in practice, and even if they are met, the dataset soon becomes outdated. Therefore, it is crucial to use several datasets for conducting algorithmic studies like ours [5, 6]. We describe the datasets used in our experiments in Section 5.

4.6 The Roadmap

In this section, we provide a "roadmap" to solving a network intrusion detection problem using anomaly detection techniques. We identify different components of the task and describe the steps to be taken. We follow this roadmap for designing our own experimental methodology.

Fundamentally, anomaly-based network intrusion detection is an anomaly detection problem which itself is a ML problem. Hence, most of the unidentified components are typical for a classic ML problem, while some are specific to anomaly detection and network intrusion detection. We have grouped the components into six blocks that are illustrated in Figure 15:

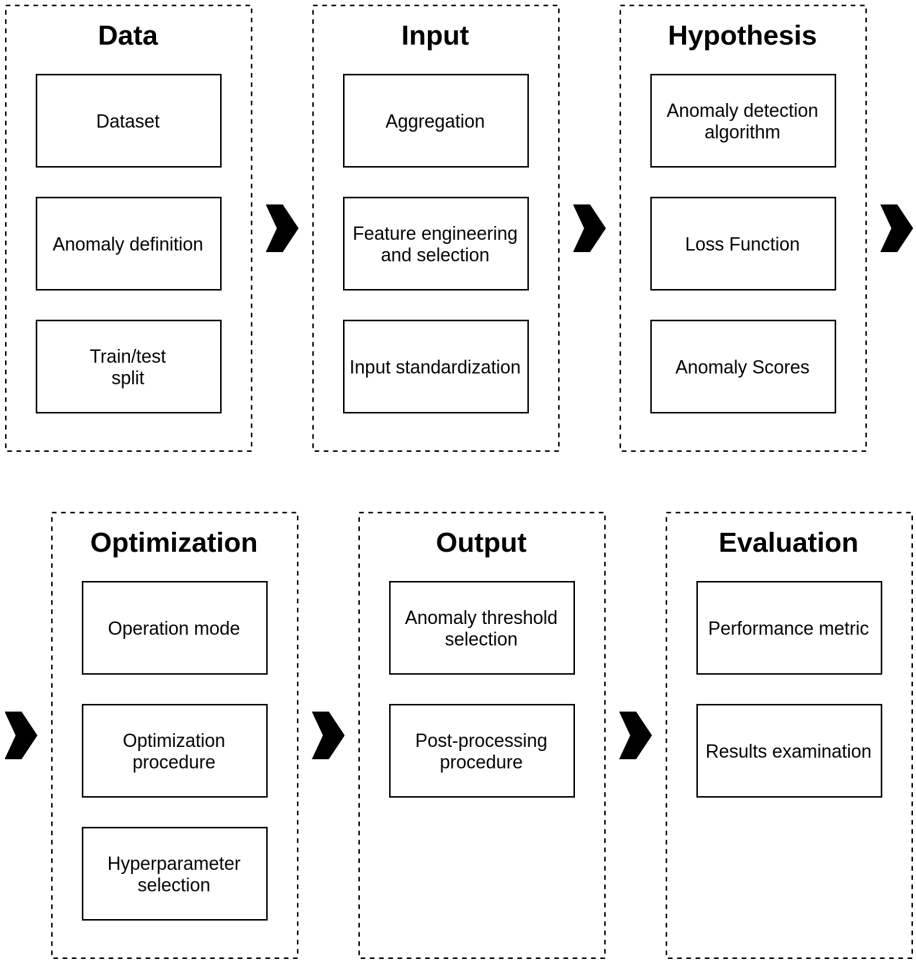


Figure 15: The roadmap to solving a network intrusion detection problem using anomaly detection

- *Data.* In this block we select the data and decide how we use it in the experiments.
  - *Dataset.* We select a network intrusion detection dataset from the publicly available ones or create a custom dataset, As discussed in Section 4.5, ideally, the selected datasets should contain traffic from a real network environment and a representative set of attacks, span a sufficient period, and have accurate labels.
  - *Anomaly definition.* We define what data points are considered to be anomalies. We have discussed various anomaly definitions in Section 3.2.1. For network intrusion detection, a common approach is to model attack traffic as anomalous and benign traffic as normal.
  - *Train/test split.* We define how to split the data into train and test sets. Public datasets may already have a predefined split, which should be preferred. Generally, we can split the data points at random or by some scenario, e.g., by date or attack type.
- *Input.* In this block we define how we preprocess the data before using it to train the model.
  - *Aggregation.* We can compute aggregated network flow statistics if the data is in the flow-based format. The details are discussed in the Section 4.4.
  - *Feature engineering and selection.* We can extract new features from the original data that better represent the underlying problem. Furthermore, we can use a special procedure, such as the wrapper method, to find an optimal subset of features. These techniques can improve the performance of the anomaly detection model.
  - *Input standardization.* We can perform input standardization to normalize the range of independent features of the data.
- *Hypothesis.* In this block, we define the hypothesis space.
  - *Anomaly detection algorithm.* We select an anomaly detection algorithm. Generally, any of the techniques discussed in Section 3.2.2 can be used; however, a few considerations should to be taken into account. Firstly, the technique should be suitable for the type of the selected data, such as medium dimensional tabular data in the case of flow-based format. Secondly, the underlying assumptions of the technique should hold for the data. Often, this can only be established empirically. Thirdly, the special requirements for computational complexity, explainability, and robustness should be taken into account.
  - *Loss function.* We choose a loss function for the optimization of our model. Often there is a standard choice, such as the mean squared error for AE

(Eq. 2). The loss functions used in our experiments are presented in Table 3.

  - *Anomaly detection score.* We choose a function to assign anomaly scores. The anomaly score function may or may not be the same as the loss function. For example, there are two options for VAE: the mean squared error (Eq. 2) or the generative probability (Eq. 4). The anomaly scores used in our experiments are presented in Table 3.

- *Optimization.* In this block, we define the optimization procedure.

  - *Operation mode.* We decide if the model should be optimized in the unsupervised, semi-supervised, or supervised mode (cf. Section 3.2.1). This depends on the model itself (not all three may be applicable), the availability of labels, the nature of the task, and so on. Generally, unsupervised or semi-supervised techniques are preferred for network intrusion detection.
  - *Optimization procedure.* We define the procedure to optimize the model. For deep learning models, we need to select an optimization algorithm, such as Adam, and specify its parameters.
  - *Hyperparameter selection.* We can experiment with different combinations of the hyperparameters of the model and the optimization algorithm. The best performing combination is typically identified on the validation batch.

- *Output.* In this block, we define how to postprocess the output of the algorithm.

  - *Anomaly threshold selection.* We define a procedure to select the anomaly detection threshold (see Section 3.2.1). For example, we can choose the best performing threshold on a hold-out batch which we refer to as the threshold batch.
  - *Postprocessing procedure.* We can design a postprocessing procedure to clean up the output of the model.

- *Evaluation.*

  - *Performance metric.* We choose a metric to evaluate the model's performance. The common choices include F_1-score and ROC AUC. However, depending on the nature of the problem, using a custom performance metric may provide a better picture.
  - *Results examination.* We can examine the results more thoroughly, such as by analyzing false positives, feature importance, and performance per different attack types. Such analysis can help us obtain insight into the model's behavior, which is often more valuable than achieving a higher value of the performance metric.

5 Datasets

We evaluate the methods on three modern network intrusion datasets: CTU-13 [7], CICIDS 2017 [8] and IoT-23 [9]. This section provides a short description of each dataset and some of its most relevant properties.

We will need some basic definitions:

- *Benign and Attack Labels.* We use the following convention for the label names: the data points that indicate some suspicious or malicious activity have the label “attack”, and the rest have the label “benign”.
- *Attack scenarios.* By an attack scenario we mean a specific case of the execution of some attack. For example, it may be a DDoS attack executed on March 14, 2016. In the following, we have identified the attack scenarios of each dataset.
- *Contamination rate:* The contamination rate, also called the anomaly ratio, is the percentage of attack data points (anomalies) in a single attack scenario or a whole data set.

5.1 Dataset Descriptions

5.1.1 CTU-13

The CTU-13 dataset [7] was created in the CTU University in 2011. The authors captured traffic from the university network and executed 13 different botnet attacks, performing various malicious actions, such as DDoS attack and port scanning [59]. Table 8 shows the characteristics of the CTU-13 attack scenarios.

The CTU-13 data is available in both packet and bidirectional network flow formats. The authors provide three types of labels: "attack", "normal" and "background". The difference between normal and background traffic is that the latter has not been manually examined and can potentially contain malicious traffic. However, in our experiments, we consider the background traffic to be normal for the reason that there are not enough normal data points alone. The dataset is publicly available.

5.1.2 CICIDS 2017

The CICIDS 2017 dataset [8] was created by the Canadian Institute for Cybersecurity. The network traffic was emulated with special software using a realistic network infrastructure. The capture spans a period of five days. The authors have executed a wide range of attacks, including SSH brute force, heartbleed, botnet, DoS, DDoS, web, and infiltration attacks. Table 9 shows the characteristics of the attack scenarios.

The CICIDS 2017 data is available in both packet and bidirectional network flow formats. The authors use a custom network flow collector that extracts more than 80 attributes for each flow. CICIDS 2017 contains normal and detailed attack labels. The dataset is publicly available.

Id	Name	#Total	#Benign	#Attack	Description
1	Neris	2824609	2824609	40961 (1.45%)	IRC, Spam, CF
2	Neris	1808104	1787163	20941 (1.16%)	IRC, Spam, CF
3	Rbot	4710409	4683587	26822 (0.57%)	IRC, PS
4	Rbot	1121062	1118482	2580 (0.23%)	IRC, UDP and ICMP DDoS
5	Virut	129830	128929	901 (0.69%)	Spam, PS, HTTP, Scan web proxies
6	Menti	558911	554281	4630 (0.83%)	PS, C&C, RDP
7	Sogou	114074	114011	63 (0.06%)	HTTP, Chinese hosts
8	Murlo	2954167	2948040	6127 (0.21%)	PS, C&C, Net-BIOS, STUN
9	Neris	2087487	1902500	184987 (8.86%)	IRC, Spam, CF, PS
10	Rbot	1309775	1203423	106352 (8.12%)	IRC, UDP DDoS
11	Rbot	107245	99081	8164 (7.61%)	IRC, ICMP DDoS
12	NSIS.ay	325466	323298	2168 (0.67%)	P2P, Synchronization
13	Virut	1925097	1885094	40003 (2.08%)	SPAM, PS, HTTP, Captcha, Web Mail

Table 8: Attack Scenarios of the CTU-13 dataset. The table specifies the number of network flows in each scenario, as well as the contamination rate which are shown in parenthesis. (CF- Click Fraud, PS - Port Scan, RDP - Remote Desktop Protocol, P2P - Peer-to-peer)

Id	Name	#Total	#Benign	#Attack	Description
1	Benign	529918	529918	0 (0.00%)	No attack
2	DDoS	225745	97718	128027 (56.71%)	LOIC DDoS
3	Port Scan	286467	127537	158930(55.48%)	NMap (sS, sT, sF, sX, sN, sP, sV, sU,sO, sA, sW, sR, sL and B)
4	Botnet	191033	189067	1966 (1.03%)	Botnet ARES
5	Infiltration	288601	288565	36 (0.01%)	Infiltration Dropbox, download and cool disk
6	Web	170366	168186	2180 (1.28%)	Web BForce, XSS and SQL Inject.
7	Brute-force	445909	432074	13835 (3.10%)	FTP-Patator, SSH-Patator
8	DoS	692703	440031	252672 (36.48%)	DoS tools: Hulk, GoldenEye, Slowloris, Slowhttptest; Heart-bleed

Table 9: Attack Scenarios of CICIDS 2017 [8] (LOIC - Low Orbit Ion Canon, XSS - Cross-Site Scripting, SQL - Structured Query Language)

5.1.3 IoT-23

IoT-23 [9] is a novel dataset created in the Stratosphere Laboratory, CTU University, in 2020. The dataset contains 23 captures of different network traffic from IoT devices, out of which 20 captures contain malware traces. In each malicious scenario, the authors execute a specific botnet malware sample on a Raspberry Pi. The benign scenarios contain traffic from three real IoT devices: a Philips HUE smart LED lamp, an Amazon Echo home intelligent personal assistant, and a Somfy smart doorlock. Table 10 shows the characteristics of the attack scenarios. Due to the limitations on computational resources, we only experiment with Mirai scenarios and limit the total number of data points to 5,000,000.

The IoT-23 data is available in packet- and bidirectional- flow format. The authors provide benign and malicious labels, including detailed information for various malicious activities, performed by the botnets:

- *C&C.* This label indicates that the infected device communicates with a C&C server.
- *Horizontal Port Scan (HPS).* This label indicates that the infected device performs a horizontal port scan
- *DDoS.* This label indicates the infected device executes a Distributed Denial of Service attack
- *Attack (ATK).* This label indicates that the infected device executes some other type of attack against another host.
- *File Download (FDL).* This label indicates that a file is being downloaded to the infected device.

The dataset is publicly available.

5.2 Other Datasets

Below, we provide a short overview of some of the other existing network intrusion datasets that are publicly available:

- *1999 DARPA IDS* [60] was created at the MIT Lincoln Lab. The data was captured in a packet-based format within an emulated network environment. The dataset contains five weeks of network traffic, including various kinds of attacks.
- *KDDCUP'99* [61], published by the University of California, Irvine, is based on the 1999 DARPA IDS dataset, and it is one of the most popular datasets for benchmarking of network intrusion detection methods [6]. The data contains both packet- and flow-level attributes, and includes more than 20 different types of attacks. Presently, the use of this dataset is frowned upon (as well as 1999 DARPA IDS), as is has been shown to have unrealistic attacks and a large amount of redundant records [6, 16].

Id	Name	#Total	#Benign	#Attack	Description
1	Benign (7)	130	130	0 (0.00%)	Sompfy Door Lock
2	Benign (4)	452	452	0 (0.00%)	Philips HUE
3	Benign (5)	1374	1374	0 (0.00%)	Amazon Echo
4	Mirai (1)	1008748	469275	539473 (53.48%)	C&C, DDoS, HPS
5	Mirai (48)	3394338	3734	3390604 (99.89%)	C&C, Attack, FDL, PS
6	Mirai (52)	5000000	405	4999595 (99.99%)	C&C, FDL, HPS
7	Mirai (44)	237	211	26 (10.97%)	C&C, FDL, DDoS
8	Mirai (34)	23145	1923	21222 (91.69%)	C&C, DDoS, HPS
9	Mirai (49)	5000000	3433	4996567 (99.93%)	C&C, FDL, HPS
10	Mirai (35)	5000000	4662273	337727 (6.75%)	C&C, FDL, DDoS

Table 10: Attack Scenarios of IoT-23 [9]. The original scenario indices are indicated in the parenthesis. (HPS: Horizontal Port Scan, FDL: File Download, C&C: Command & Control communication/activities)

- *NSL-KDD* [16] is an enhanced version of the KDDCUP'99 dataset. The authors removed duplicate data points from the complete KDDCUP'99 dataset and provided more balanced train and test subsets.
- *UGR-16* [10], published by Network Engineering & Security Group (NESG), contains four months of data captured from an internet service provider environment. The authors executed several attacks (DoS, botnet, and port scans) and mixed them with the traffic from the real environment.
- *UNSW-NB15* [26], published by the Australian Centre for Cyber Security, contains normal and malicious network traffic created using the IXIA Perfect Storm tool in a realistic network testbed. The executed attacks include backdoors, DoS, exploits, fuzzers, and worms.

5.3 Overview

In what follows, we summarize the properties of the selected datasets and argue why they constitute a proper context for a general evaluation of anomaly detection techniques. Table 11 provides an overview of the selected datasets with respect to the common properties identified in Section 4.5.

Firstly, the datasets contain traffic from three different network environments: university network traffic, emulated traffic, and IoT traffic. This diversity allows us to avoid overfitting to a particular environment when making conclusions, but at the same time, it allows us to identify the most promising use cases. Secondly, for all three datasets, the provided traffic is available in the bidirectional network flow format, which essentially allows us to apply the same methods to them. Thirdly, each dataset contains a wide range of executed attacks and the corresponding detailed

labels. As we have already pointed out, obtaining accurately labeled data is crucial for evaluating anomaly detection methods.

Lastly, we should mention that any of the datasets presented in Section 5.2 also could have been used in the study, but were not included due to the computational and time limitations.

Dataset	**Year**	**Format**	**Size**	**Duration**	**#Attacks**	**Type of Network**	**Predef. Split**
CTU-13	2013	packet, bi.flow	2.6GB flows	5 days	13	university	yes
CICIDS 2017	2017	packet, bi.flow	1.2GB flows	5 days	7	emulated	no
IoT-23	2020	packet, bi.flow	60GB flows	20 days	20	IoT	no

Table 11: Overview of the selected network intrusion datasets

6 Experiments

In this section, we present the experimental part of our work, in which we have conducted a thorough comparative study of anomaly detection techniques for network intrusion. Section 6.1 provides a detailed description of the experiment setup, which includes the processing and splitting of the data, the algorithm setup, and the evaluation methodology. Section 6.2 presents the obtained results and the main findings, which are further discussed in Section 6.3.

6.1 Experiment Setup

In this section, we present our experiment methodology. In essence, our goal is to compare five different anomaly detection algorithms (AE, VAE, AEGMM, IF, and MAH, presented in Section 3.2.3) trained in the semi-supervised mode on the network intrusion data from three sources (CTU-13, CICIDS 2017, and IoT-23, presented in Section 5), with and without network flow aggregation. For every possible combination of the algorithms and data sources, we have carried out an experiment consisting of training a model, selecting the optimal detection threshold, and evaluating its performance on a test set. For designing our methodology, we followed the steps from the roadmap (presented in Section 4.6)

6.1.1 Benchmark Data

To evaluate the selected algorithms in a realistic way, we use the data from the original sources to create appropriate anomaly detection benchmarks. As previously mentioned, we use data in the bidirectional network flow format.

Firstly, we need to provide the definition for anomaly. Here we make the common choice of modeling benign network traffic as normal and attack traffic as anomalous:

$$\text{benign} = \text{normal}, \;\; \text{attack} = \text{anomalous}$$

The next step is to preprocess the original data, which includes inferring the missing values, parsing TCP flags and other attributes. For each dataset, we extract a set of network flow features as specified in Appendix A. For categorical features , such as *protocol*, we use One-Hot Encoding (OHE). We encode binary features, such as *fwd_ack_flag* (a TCP ACK flag sent from the source to the destination node), using a single binary column.

With regard to our benchmarks, Figure 16 illustrates our methodology on the CTU-13 example. As can be seen from the figure, for each original dataset we obtain six different versions of benchmark datasets. There are two stages in the process: aggregation of network flows and splitting the data into train and test sets. We have already discussed the idea behind network flow aggregation in Section 4.4, and we provide the details for our aggregation method in Section 6.1.2. We use two different aggregation windows: a one-minute window and a three-minute one. Thus, after the first stage of aggregation, we have three versions of the data: the original network

flows (denoted as NO_AGGR), and the aggregations of the flows over one minute (AGGR_1M) and over three minutes (AGGR_3M).

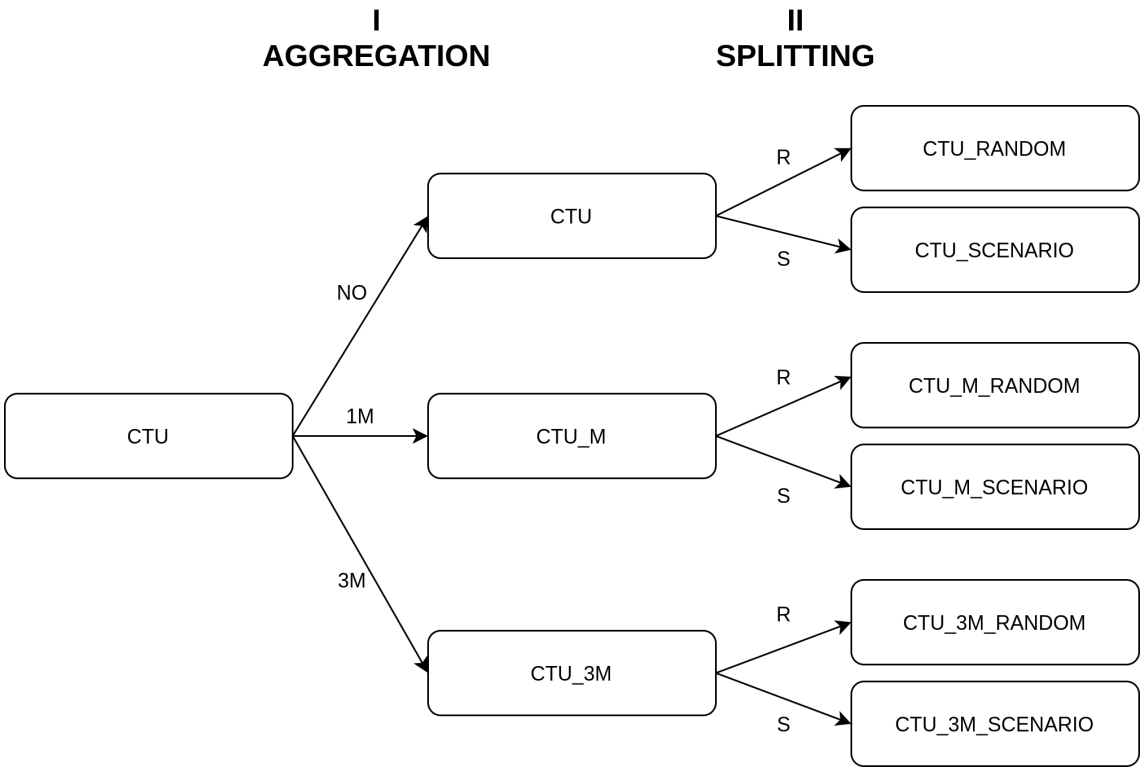


Figure 16: Creating benchmark datasets from the original data. (*Stage I Aggregation*: NO - no aggregation, 1M - aggregation over a one-minute window, 3M - aggregation over a three-minute window; *Stage II Splitting*: R - splitting into train/test at random, S - splitting into train/test by attack scenario).

The second stage is separating the data into train and test splits. Here we use two different strategies:

- *Split-at-random.* As the name suggests, we separate the data into train and test splits at random. First, we separate all the data points into normal and anomalous. We randomly take 80% of the normal data and 20% of the anomalous data for training, and randomly shuffle the data points. Conversely, we randomly take 20% of the normal data and 80% of the anomalous data for testing, and randomly shuffle the data points. The logic behind the choice of percentages is motivated by the fact that we want to ensure a significant presence of anomalies in the test set, given the existing class imbalance. As we train the models the semi-supervised mode, we only use the anomalies in the training set for hyperparameter tuning and detection threshold selection. On the other hand, both anomalous and normal data points are used during testing. Thus, we believe it is beneficial for an accurate evaluation to place a larger percentage of the anomalies into the test split.

- *Split-by-scenario.* In this case, we split the data points by attack scenarios. The attack scenarios for each dataset have been discussed in Section 5). Table 12 presents our choice of the training and testing splits for each dataset. For CTU-13, we have used the split suggested in the original paper [7]. For CICIDS 2017 and IoT-23, there are no predefined splits; hence, we have made the choice ourselves.

Dataset	**Scenario ID**	
	Train	**Test**
CTU-13	3, 4, 5, 7, 10, 11, 12, 13	1, 2, 6, 8, 9
CICIDS 2017	1, 2, 3, 6	4, 5, 7, 8
IoT-23	1, 2, 3, 4, 6, 7	5, 8, 9, 10

Table 12: The selected train/test scenario splits for each dataset. The scenarios for each dataset are described in Tables 8, 9 and 10. We identify the scenarios by their IDs, that we assign ourselves.

We hypothesize that the split-by-scenario case poses a more difficult task for the models, as they do not experience either normal or anomalous data points from any of the test scenarios. While the benign traffic of the test scenarios is likely to resemble that of the train scenarios, the attack traffic may exhibit a completely different nature, which might make it harder to detect. In this way, we can test how good the models are at detecting unknown attacks in a more realistic fashion.

The last step in the benchmark creation is the construction of train, threshold and test batches (Table 13):

- *Train Batch.* This batch is used for the training of the models. It contains only normal data points that are taken from the train split.

- *Threshold Batch.* This batch is used for hyperparameter tuning and detection threshold selection. It has a fixed size of 10000 data points and a fixed contamination rate of 5%. We have found that having a fixed percentage of anomalies helps with finding a good detection threshold. The normal and anomalous data points of the threshold batch are sampled from the train split. We sample the normal data points uniformly, but we sample the anomalies "fairly", i.e. in such a way that each attack is represented in an equal proportion. The reason why is that the number of available data points per attack type varies greatly, e.g., it is substantial for a DDoS attack, but small for C&C communication. We stress that only the data points from the train split are taken; in the split-by-scenario case, the attacks from the testing scenarios don't get selected into the threshold batch.

- *Test Batch.* The test batch is used for the final evaluation of the models.

In the split-at-random case, we take all the normal data points from the test split, but we re-sample the test anomalies (fairly) so that they constitute 10 % of the batch. We found that fixing the contamination rate of the test batch produces more interpretable results and allows us to compare the models in a more meaningful way.

In the split-by-scenario case, we take all the normal and anomalous data points *as is*, preserving the original contamination rate, as well as the total number of data points.

Batch	**Split**	
	Random	**Scenario**
Train	• all normal from train split • no anomalous	• all normal from train split • no anomalous
Threshold	• 9500 normal from train split • 500(5%) anomalous from train split	• 9500 normal from train split • 500(5%) anomalous from train split
Test	• all normal from test split • 10% anomalous from test split	• all normal from test split • all anomalous from test split

Table 13: Construction of train, threshold and test batches. The percentages are given with respect to the total batch size. If the percentage of anomalous data points is fixed, they are fair-sampled (with replacement if there are not enough points).

To summarize, the flow of a single experiment is as follows: given a benchmark dataset as input, a model is trained on the train batch, the optimal values of detection threshold and hyperparameters are tuned on the threshold batch, and the model is evaluated on the test batch.

6.1.2 Aggregated and Network Flow Features

In this section, we explain our method for aggregating network flows. The motivation for performing this operation was given in Section 4.4. Finally, for each dataset we identify the set of features used to train the models for both NO_AGGR and AGGR cases.

Recall that we group network flows by source IP address, and for each group we calculate a set of descriptive statistics over a sliding aggregation window of fixed width. The width should be long enough to capture the attack patterns and short enough not to capture too much traffic. The optimal width of the window is specific to each dataset, so it is a good idea to try several values. In this work, we use two

aggregation windows for each dataset: a one-minute window (AGGR_1M) and a three-minute one (AGGR_3M).

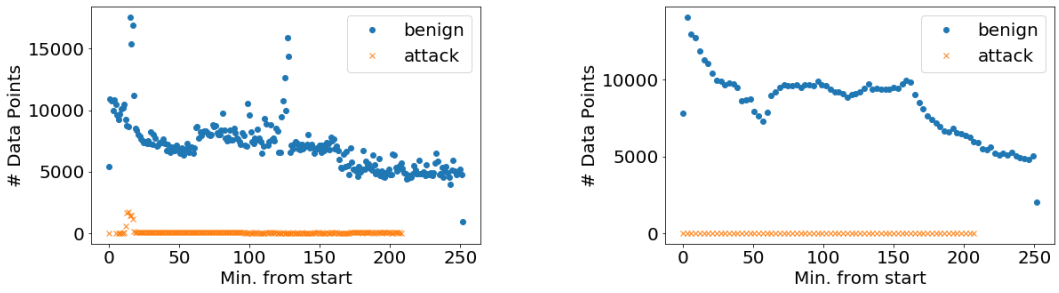
We calculate the descriptive statistics in the following way. For numerical features, we calculate the mean, maximum, minimum, median, and standard deviation (SD). For example, for the network flow feature *tot_byts*, the total number of bytes exchanged in a network flow, we obtain five new features: *tot_byts_mean*, *tot_byts_max*, *tot_byts_min*, *tot_byts_median*, and *tot_byts_std*. For categorical features, we calculate the number of unique values and the information entropy, $H(X) = -\sum_i P_X(x_i) \log P_X(x_i)$. For example, for the list of destination IP addresses, we obtain two new features - *dst_ip_num_uniq* and *dst_ip_entropy*. Lastly, we need to explain how to infer new labels for the aggregated data points. We use the following rule: we assign the "attack" label if at least one flow in the group was labeled "attack".

Table 14 shows the number of features used for training for both AGGR and NO_AGGR data. Due to space limitations, the complete listings of features for each dataset are provided in Appendix A. There are a few considerations to point out. Firstly, we do not calculate the described aggregated statistics for every original network flow feature, but only for those features that we believe are important for detection based on our expert knowledge. Secondly, the IP address and port information is discarded in the NO_AGGR case, as there is no simple way to encode it (see Section 4.3). On the other hand, in the AGGR case we extract new numerical features from the IP addresses and ports, such as *dst_ip_num_uniq* - number of unique destination IP addresses, and *dst_ip_entropy*, the entropy of destination IP addresses.

Dataset	**Num. of features**	
	NO_AGGR	**AGGR**
CTU-13	25	32
CICIDS 2017	76	45
IoT-23	37	57

Table 14: The number of features for each dataset

Lastly, we demonstrate how performing network flow aggregation reduces the total number of data points and changes the contamination rates. Figure 17 illustrates the effect of aggregating network flows on the example of Scenario 2 from the CTU-13 dataset. Due to space limitations, we present the rest of the attack scenarios of each dataset in Appendix B. Table 15 shows the new number of benign and attack data points for each dataset after aggregating network flows. It is important to realize that aggregation reduces the contamination rate for certain attack types, like DDoS and port scan attacks, where large number of flows is emitted in short bursts of time. This is most noticeable in Scenarios 5 and 6 from the IoT-23 dataset.



(a) No aggregation (NO_AGGR) (b) With aggregation (AGGR_3M)

Figure 17: Transforming data points using network flow aggregation on Scenario 2 (Neris), CTU-13

Dataset	NO_AGGR		AGGR_1T		AGGR_3T	
	# Benign	# Attack	# Benign	# Attack	# Benign	# Attack
CTU-13	19531537	444699 (2.23%)	5944746	5608 (0.09%)	5567856	2140 (0.04%)
CICIDS	2273096	557646 (19.70%)	194278	1006 (0.52%)	147282	443 (0.30%)
IoT-13	5143210	14285214 (73.53%)	24993	8736 (25.90%)	23597	2926 (11.03%)

Table 15: Reducing the number of data points using network flow aggregation (contamination rates are shown in parenthesis)

6.1.3 Aggregation of Anomaly Scores

In this section, we propose a simple post-processing method that can be used to clean up the predictions of models trained on NO_AGGR data. We apply the same idea of aggregation to the output of these models, which is anomaly scores assigned to individual network flows.

Concretely, we again group network flows by a unique source IP and over an aggregation time window. Our goal is to assign an anomaly score to each group. However, we only have the anomaly scores assigned to network flows by the NO_AGGR model. We infer the group anomaly score in the following way - choose the maximum anomaly score of all the flows in the group. We infer the ground truth labels in the same way as before - assign the "attack" label if at least one flow the group was labeled as "attack".

On a high level, the difference between aggregating network flows and aggregating anomaly scores is as follows. In the former case, we perform the aggregation operation first, compute a new set of features, and use them to train models and predict anomaly scores. In the latter case, we use the original network flow features to train the models and predict anomaly scores, and afterwards aggregate the scores. Importantly, if aggregation window of the same width are used, both methods produce anomaly scores for the same aggregated data points, which allows us to compare them directly.

6.1.4 Algorithm Setup

A brief theoretical description of the selected algorithms, as well as a discussion on their suitability for network intrusion detection have been provided in Section 3.2.3. We have implemented our framework for training and evaluating models in `Python 3.0`, using a novel anomaly detection library `alibi-detect` [53]. For the deep autoencoding models, we used the `tensorflow` backend, and for the classic ones - `sklearn`.

For training of the deep autoencoding models, the Adam optimizer [62] is used with a learning rate of 0.001. We train each model for a total of 5 epochs (limiting each epoch to at most 500 steps) using different values of batch sizes. The specific configuration parameters of each deep model are summarized below:

- *AE:* We try different configurations for the encoder and decoder networks of an autoencoder. The number and size of layers of a decoder network architecture always mirror that of the corresponding encoder network. As an example, let the input dimensionality be $input\_dim = 30$ and the dimensionality of the compact representation be $encoding\_dim = 3$. Suppose, the encoder runs with FC(30, 20, ReLU) - FC(20, 10, ReLU) - FC(10, 10, ReLU) - FC(10, 3, None). Then, the decoder architecture runs with FC(3, 10, ReLU) - FC(10, 10, ReLU) - FC(10, 20, ReLU)- FC(20, 30, None), mirroring the encoder. In all experiments, we use ReLU as a non-linearity activation function, and we do not use any non-linearity activation function for the last layers of the decoder and encoder. Naturally, the output dimension of the decoder is the same as the input dimension. For brevity, we denote the above architecture as $encoder\_net$ = 20-10-10 and $encoding\_dim = 3$. The input and output dimensions depend on the dataset, and the rest can be inferred as described above.

- *VAE:* The VAE architecture is conceptually similar to that of the AE, except it has two latent variables. An example of the VAE encoder is FC(30, 20, ReLU) - FC(20, 10, ReLU) - FC(10, 5, ReLU) - FC(5, 1, None)x2. This encoder turns the input into two variables (the mean and variance of the modelling probability distribution) in the latent space, that has the dimension $latent\_dim = 1$. In this example, the decoder will run with FC(1, 5, ReLU) - FC(5, 10, ReLU) - FC(10, 20, ReLU) - FC(20, 30, None). We denote the above example as $encoder\_net$ = 20-10-5 and $latent\_dim = 1$.

- *AEGMM:* The compression network of the AEGMM is exacly the same as the basic AE. An example architecture is FC(30, 20, ReLU) - FC(20, 10, ReLU) - FC(10, 1, None) - FC(1, 10, ReLU) - FC(10, 20, ReLU) - FC(20, 30, None), where $encoding\_dim = 1$. We use two functions for computing the reconstruction error: euclidean distance and cosine similarity. We fix the GMM network to be FC(1 + 2, 10, tanh)-Drop(0.5)-FC(10, 4, softmax) - FC(3, 10, tanh)-Drop(0.5)-FC(10, 4, softmax) (the same one as in the original paper [13]) where the last dim is num_gmm, the number of GMM components. We

experiment with different compression network architectures and number of GMM components. We use the batch size of 2048, as we found that a larger batch size is beneficial for optimization.

where FC(a, b, f) means a fully-connected layer with a input neurons and b output neurons activated by function f (None means no activation function is used), and Drop(p) denotes a dropout layer with keep probability p during training.

The configuration parameters of the classic models are summarized below:

- *IF:* We experiment with different values of *num_estimators* , the numbers of isolation trees used in the ensemble.
- *MAH:* We use an online implementation of the algorithm and perform the dimensionality reduction before computing the Mahalanobis distance. We experiment with values of *num_components*, the number of PCA components. Since MAH is the only online method, we also use a different training procedure: instead of optimizing on the train batch, we "warm up" the model on the threshold batch. Hyperparameter tuning and detection threshold selection are done in the same way as for the other models.

For brevity, we use the following convention when referring to the models. First, we specify the algorithm used, then if it was trained on the network flow feature set or aggregated feature set, and lastly, the source dataset. For example, by VAE AGGR_1M IoT-23 we mean a VAE model trained on the aggregated network flows from the IoT-23 dataset. Sometimes we are going use broader terms, e.g., the AGGR models to specify all the models trained on the aggregated flow data.

6.1.5 Hyperparameter and Threshold Selection

In this section, we explain our methodology for selecting the optimal combination of hyperparameters together with an anomaly detection threshold.

We experiment with different architectures of the deep-learning-based autoencoding models, as well as different configurations of classic models. Table 16 presents the tested hyperparameter settings for each algorithm. For each combination of hyperparameters, we train a model on the train batch and evaluate it on the threshold batch, after selecting a detection threshold as described below. Finally, we select the combination that achieves the highest performance score on the threshold batch.

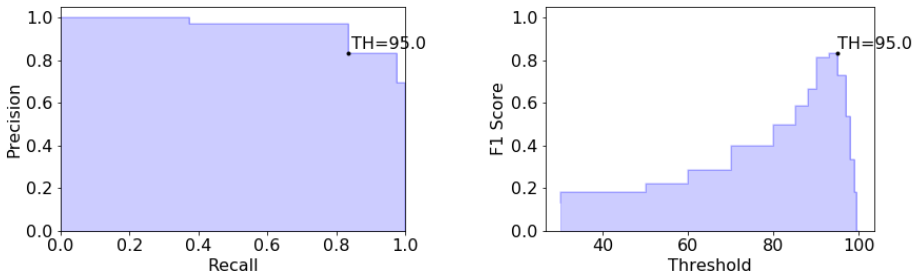
Figure 18 illustrates the evaluation process on the threshold batch. Given anomaly scores assigned by the model, we plot a precision-recall curve and select the detection threshold that maximizes the F_1-score. By our design, the threshold batch always has a fixed contamination rate of 5%; therefore, a well-performing model should select a threshold value close to 95%.

6.1.6 Evaluation Methodology

We perform the evaluation of the models on the test batch using the F_1-score as our main performance metric. However, we also examine precision and recall separately. The definitions of each metric are provided below:

Model	Parameter	Dataset	
		CTU-13	**CICIDS 2017, IoT-23**
AE	*encoder_net*	{ 15-5, 20-10-5, 20-10-10-5 }	{ 50-10, 60-30-10 }
	encoding_dim	3	3
	batch_size	{64, 1024 }	{64, 1024 }
	num_epochs	5	5
VAE	*encoder_net*	{ 15-5, 20-10-5, 20-10-10-5 }	{ 50-10, 60-30-10 }
	latent_dim	{ 1, 2, 5 }	{ 1, 2, 5 }
	batch_size	{64, 1024 }	{64, 1024 }
	num_epochs	5	5
AEGMM	*encoder_net*	{ 15-5, 20-10-5, 20-10-10-5 }	{ 50-10, 60-30-10 }
	encoding_dim	1	1
	num_gmm	{ 2, 4 }	{ 2, 4 }
	batch_size	2048	2048
	num_epochs	5	5
IF	*num_estimators*	{ 50, 100, 200 }	{ 50, 100, 200 }
MAH	*num_components*	{ 2, 4, 8 }	{ 2, 4, 8 }

Table 16: Tested hyperparameter settings. For each model, we try every possible combination of the hyperparameters.



(a) Precision-recall curve (b) F1-score curve

Figure 18: Selecting the optimal threshold on the threshold batch

- *Precision*, which answers the question of "how many selected instances are true attacks?":

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{11}$$

where TP stands for true positives (attack instances predicted correctly) and FP stands for false positives (benign instances predicted as attack).

- *Recall*, which answers the question of “how many true attack instances are selected?”:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{12}$$

where TP stands for true positives and FN stands for false negatives (attack instances predicted as benign).

- F_1-*score*, which is the harmonic mean of the precision and recall:

$$F_1 - \text{score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{13}$$

The F_1-score is often used to measure performance for problems with class imbalance, i.e., when one class has significantly more examples than other classes. This metric is suitable for us, as in all but one of our benchmarks the attack class is the minority class (see Table 15). The only exception is IoT-23 NO_AGGR that has a contamination rate higher than 70 %; in this case, we invert the definition of the positive class when computing the F_1-score. That means that true positives are benign instances predicted correctly, false positives are attack instances predicted as benign, and false negatives are benign instances predicted as attack. Keeping the original definition of F_1-score (with attacks being the positive class) is not very informative, as predicting all test data points as attacks achieves a high F_1-score.

For comparison of different algorithms on multiple datasets, we follow the statistical procedure described in [63]. Based on the obtained test F_1-score values, we rank each algorithm according to its performance (from 1 to 5, with 1 being the best-performing rank) and average the ranks over all benchmark datasets. Then, we conduct two statistical tests: the Friedman test which answers the question of "do all the algorithms perform equally well?", and Nemenyi test which answers the question of "do two algorithms perform differently on a statistically significant level?".

The Friedman statistic is given by:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right], \tag{14}$$

where N is the number of datasets, k is the number of algorithms, R_j is the average rank of the j-th algorithm. In our case (5 algorithms and 9 benchmark datasets) the critical value for the Friedman test at 5% confidence level is $q_{0.05}^F \approx 9.244$.

The Nemenyi test determines that the performance of two algorithms is significantly different if the corresponding average ranks differ by at least the critical difference:

$$CD = q_\alpha^N \sqrt{\frac{k(k+1)}{6N}}, \tag{15}$$

where q_α^N is the critical value at the confidence level α. In our case, the critical value for the Nemenyi test at 10% confidence level is $CD \approx 2.245$.

The results of the comparison can be visually represented with a critical difference diagram (see e.g. Figure 23). The diagram contains the average algorithm ranks and connects the statistically equally performing algorithms with a thick black line.

Furthermore, to get a better understanding of the detection capabilities of our models, we use the following two techniques:

- *Performance per attack scenario.* In this case, we examine how efficient the models are at detecting different types of attacks present in the data. We split the test data points by their corresponding attack scenario, and evaluate the performance for each group separately. We define the *detectability* of an attack as follows: *good* - 85% $<$ F_1-score $\leq$ 100%; *moderate* - 65% $<$ F_1-score $\leq$ 85%; *poor* - F_1-score $\leq$ 65%.

- *Anomaly instance score plots.* An anomaly instance score plot is a useful tool for visually evaluating how well the model separates the benign and attack data points. For contiguous network traffic data, we plot the timestamps of data points on the x-axis and their corresponding anomaly scores, assigned by the model, on the y-axis. We also plot the selected detection threshold with a black line; all the data points above the threshold are predicted as anomalies. Figures 20 and 21 show examples of anomaly instance score plots.

6.2 Experimental Results

We report the results separately for the benchmark datasets created by splitting data at random and by scenario. For each dataset, we compare the performance of the models and identify detectability of the attack scenarios, and we examine some successful and unsuccessful cases. In addition, we show how aggregating anomaly scores assigned by the NO_AGGR models can produce similar results to the scores assigned by the AGGR models. We conclude the section with a discussion of the results.

6.2.1 Runtime

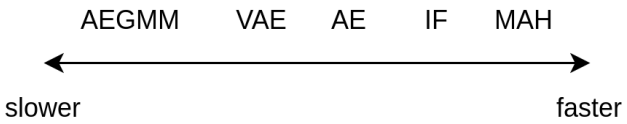
Firstly, we assess the computational complexity of the models. Figure 19 presents a relative ranking of the tested algorithms by their fit and predict time, ordering them from slow to fast. The autoencoding models are the slowest to fit, but are the fastest at predict time. The online implementation of MAH only requires a warm up on the threshold batch which makes it the fastest method on the fit time chart; however, we find it to be the slowest at predict time. Overall, all the tested methods have running times suitable for the use in a real-world NIDS.

6.2.2 Split-at-Random Benchmarks

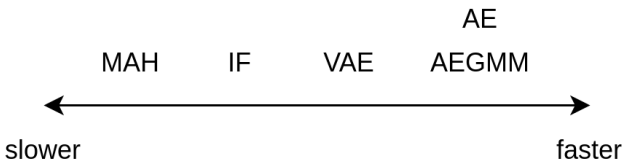
In this section, we analyze the performance of the models trained on the benchmarks split at random.

We report the best results obtained on the threshold and test batches. The former can be thought of as validation performance, while the latter tells us if the model just got lucky or cheated on the threshold batch. Due to space limitations, we specify the selected hyperparameter configurations for each model in Appendix C.

Table 17 presents the precision-recall (PR) curves achieved by each model on the threshold batch. As we can see, for all three datasets, the AGGR_T and AGGR_3T models get much closer to the top right corner (precision = 1, recall = 1) than the



(a) Fit time



(b) Predict time

Figure 19: Relative ranking of the algorithms by their fit and predict times

NO_AGGR models. Furthermore, the AGGR_3T models seem to outperform the AGGR_T models.

Table 18 presents the values of precision, recall, F_1-score and selected threshold obtained by each model on the test batch. Keep in mind that, by our design, the threshold batch has a fixed contamination rate of 5%; therefore, the correct detection threshold value is always 95% percent. The selected threshold tells us if the model achieves a high precision or recall by virtue of performing well or just making a tradeoff.

The results from Table 18 suggest that:

- Overall, we observe comparable F_1-scores for CTU-13 and CICIDS 2017. This is somewhat surprising, as we expected to see better performance on CICIDS 2017 with its emulated network traffic. We observe the best performance results for IoT-23. We believe this is because IoT devices generally have a simpler network profile. If we compare the performance between the threshold and test batches, we can see that they are, in fact, higher on the latter. This can be explained by the fact that the test batch has a higher percentage of anomalies (10% vs 5%), which seems to help the models achieve higher precision values.

- As hinted by the PR curves, for all three datasets, we observe a significant improvement in performance for the AGGR models compared to the NO_AGGR models. This is most noticeable for the IoT-23 dataset. The numbers seem to suggest that three-minute aggregation yields better results than one-minute aggregation. However, this may be affected by the fact that there are fewer data points. Another interesting observation is that the AGGR models are also more successful at selecting the detection threshold (the values are closer to the true 95%).

- The autoencoding models, in particular VAE and AE, seem to consistently outperform IF and MAH in almost all cases, although the improvement is not always significant. On the other hand, AEGMM shows inconsistent performance, ranging from the best-performing model to the worst. In a few cases, the model

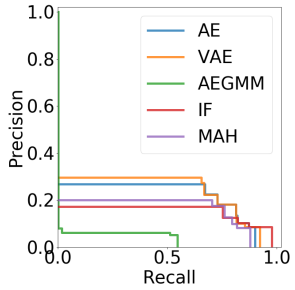
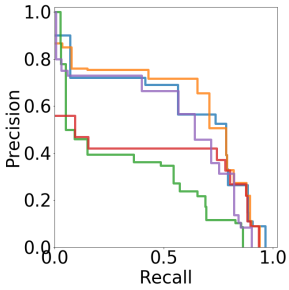
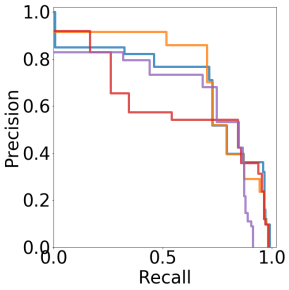
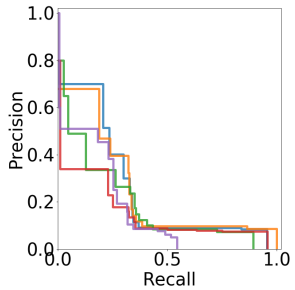
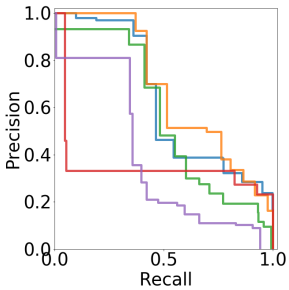
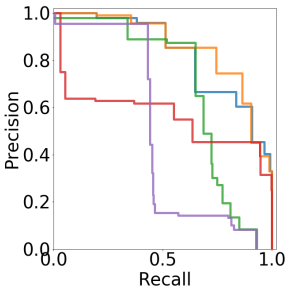
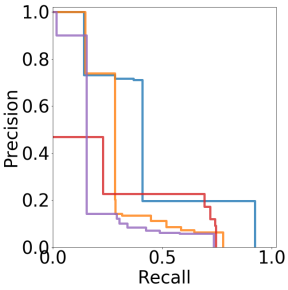
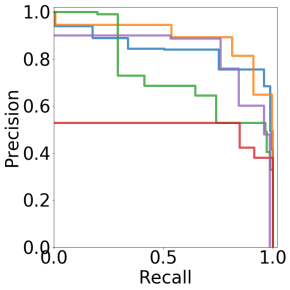
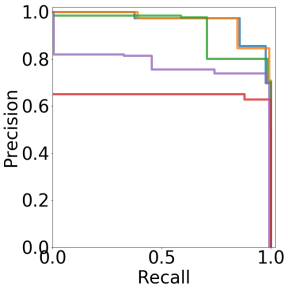
	NO_AGGR	AGGR_T	AGGR_3T
CTU-13			
CICIDS			
IoT-23			

Table 17: Precision-recall curves obtained on the threshold batches. For each model we choose the detection threshold that maximizes the F_1-score on the threshold batch.

THRESHOLD BATCH

CTU-13	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	88	0.268	0.644	0.379	93	0.526	0.736	0.613	95	**0.712**	0.712	**0.712**
VAE	90	**0.297**	0.594	**0.396**	95	**0.654**	0.654	**0.654**	95	0.702	0.702	0.702
AEGMM	60	0.062	0.498	0.111	93	0.347	0.486	0.405	-	-	-	-
IF	80	0.174	**0.694**	0.278	90	0.371	**0.742**	0.495	93	0.543	**0.76**	0.633
MAH	85	0.195	0.586	0.293	95	0.566	0.566	0.566	95	0.682	0.682	0.682

CICIDS	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	95	0.3	0.3	0.3	97	0.7	0.42	0.525	93	0.605	0.836	0.702
VAE	95	0.322	0.322	**0.322**	93	0.497	0.696	**0.58**	95	0.744	0.744	**0.744**
AEGMM	93	0.236	**0.33**	0.275	97	0.686	0.41	0.513	97	0.875	0.518	0.651
IF	97	0.34	0.204	0.255	88	0.333	**0.798**	0.469	90	0.452	**0.904**	0.603
MAH	97	**0.383**	0.23	0.287	98	**0.81**	0.324	0.463	98	**0.955**	0.382	0.546

IoT-23	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	97	0.71	0.368	**0.485**	93	0.686	0.96	0.8	95	**0.856**	0.856	**0.856**
VAE	98	0.71	0.284	0.406	95	**0.814**	0.814	**0.814**	95	0.846	0.846	0.846
AEGMM	-	-	-	-	90	0.492	**0.966**	0.652	93	0.703	**0.984**	0.82
IF	85	0.227	**0.682**	0.341	93	0.529	0.74	0.617	93	0.629	0.878	0.733
MAH	99	**0.77**	0.154	0.257	95	0.76	0.76	0.76	93	0.699	0.978	0.815

TEST BATCH

CTU-13	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	88	0.434	0.651	0.521	93	0.706	0.688	**0.697**	95	0.821	0.637	0.717
VAE	90	**0.463**	0.598	**0.522**	95	**0.797**	0.596	0.682	95	0.824	0.64	**0.721**
AEGMM	60	0.122	0.504	0.196	93	0.523	0.466	0.493	-	-	-	-
IF	80	0.307	**0.708**	0.428	90	0.563	**0.793**	0.659	93	0.705	**0.689**	0.697
MAH	85	0.303	0.666	0.416	95	0.569	0.571	0.57	95	**0.862**	0.574	0.689

CICIDS	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	95	0.455	0.298	0.36	97	0.819	0.447	0.578	93	0.744	0.758	**0.751**
VAE	95	**0.469**	0.312	**0.375**	93	0.689	0.742	**0.715**	95	0.843	0.669	0.746
AEGMM	93	0.376	**0.325**	0.349	97	0.821	0.426	0.561	97	0.91	0.36	0.516
IF	97	0.465	0.184	0.263	88	0.496	**0.744**	0.595	90	0.601	**0.826**	0.696
MAH	97	0.328	0.296	0.311	98	**0.839**	0.37	0.513	98	**0.991**	0.312	0.475

IoT-23	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	97	0.845	0.408	**0.55**	93	0.828	**0.975**	**0.896**	95	0.891	0.815	0.851
VAE	98	0.851	0.332	0.478	95	0.903	0.836	0.868	95	0.91	0.771	0.835
AEGMM	-	-	-	-	90	0.668	0.926	0.776	93	0.831	**0.956**	**0.889**
IF	85	0.408	**0.748**	0.528	93	0.713	0.789	0.749	93	0.761	0.855	0.805
MAH	99	**0.906**	0.188	0.311	95	**0.913**	0.793	0.849	93	**0.919**	0.758	0.831

Table 18: Performance on the benchmark data split at random (TH: threshold, PRE: precision, REC: recall, F1: F_1-score). Best values highlighted in bold.

has failed to converge (blank in the table). We discuss the possible reasons in Section 6.3.

In the next step of our analysis, we examine the performance of the models per attack scenario. Due to space limitation, we only discuss the key findings here, providing the complete performance results in Appendix D.

- **CTU-13.** Table 19 presents detectability of the CTU-13 attack scenarios. Overall, all but one scenario have good or moderate detectability. One interesting observation is that the best performing methods are different in each case, and none of the models excel at detecting every single scenario. For example, the AE AGGR_3M model is the best at detecting Scenario 6, Murlo; however, it completely fails at detecting Scenario 7, Sogou.

Id	Name	Detectability	Best F_1	Alg.	Aggr.
1	Neris	Good	0.933	MAH	AGGR_3M
2	Neris	Good	0.9386	MAH	AGGR_3M
3	Rbot	Moderate	0.7329	VAE	AGGR_1M
4	Rbot	Moderate	0.686	VAE	NO_AGGR
5	Virut	Good	0.9891	IF	AGGR_3M
6	Menti	Good	0.9671	AE	AGGR_3M
7	Sogou	Moderate	0.732	AEGMM	NO_AGGR
8	Murlo	Poor	0.5494	VAE	NO_AGGR
9	Neris	Good	0.9035	MAH	AGGR_3M
10	Rbot	Moderate	0.7575	VAE	NO_AGGR
11	Rbot	Good	0.9706	VAE	NO_AGGR
12	NSIS.ay	Poor	0.6418	AEGMM	NO_AGGR
13	Virut	Good	0.8842	VAE	AGGR_3M

Table 19: Detectability of the CTU-13 attack scenarios

- **CICIDS 2017.** Table 20 presents detectability of the CICIDS 2017 attack scenarios. Again, we can see that all scenarios have good or moderate detectability, and there is a lot of variance among the best performing models. However, for CICIDS all the best performing models are AGGR. A somewhat surprising finding is that the infiltration and web attacks are detected efficiently, as they are commonly believed to have poor detection capability based only on flow information [4]. These results should be interpreted with caution as they might be caused by some artifacts in the data or the insufficient amount of data.

- **IoT-23.** In case of IoT-23, instead of attack scenarios, we analyze detectability of the malicious activities performed by the botnets (see Section 5.1.3). Table 21 indicates that all but one malicious activity has good detectability. Similarly to CICIDS 2017, the AGGR models show better results than NO_AGGR.

Id	Name	Detectability	Best F_1	Alg.	Aggr.
2	DDoS	Good	0.988	AE	AGGR_1M
3	Port Scan	Moderate	0.6581	MAH	AGGR_1M
4	Botnet	Moderate	0.7073	IF	AGGR_3M
5	Infiltration	Good	0.9404	VAE	AGGR_3M
6	Web	Moderate	0.8108	AE	AGGR_3M
7	Brute-force	Good	0.8857	VAE	AGGR_3M
8	DoS	Good	0.8669	VAE	AGGR_3M

Table 20: Detectability of the CICIDS 2017 attack scenarios

Name	Detectability	Best F_1	Alg.	Aggr.
C&C	Good	0.9314	IF	AGGR_3M
Horizontal Port Scan	Good	0.9772	AE	AGGR_3M
DDoS	Good	0.9856	VAE	NO_AGGR
Attack	Moderate	0.6842	IF	AGGR_3M
File Download	Good	1.0	MAH	AGGR_3M

Table 21: Detectability of the IoT-23 malicious actions

6.2.3 Split-by-Scenario Benchmarks

In this section, we analyze the performance of the models trained on the benchmarks split by scenario.

As discussed above, this setting is harder and more realistic, as the original contamination rate is preserved in the test batch. Furthermore, the models never experience test attacks during training, which allows us to estimate how good they are at detecting unseen attacks. As in the split-at-random case, we perform the hyperparameter and threshold selection on the threshold batch. Due to space limitations, we specify the selected hyperparameter configurations for each model in Appendix C.

Table 22 presents the values of precision, recall, F_1-score, and selected threshold obtained by each model on the test batch.

The results from Table 22 suggest that:

- Overall, the performance on the threshold batch is comparable to the split-at-random case. Interestingly, the performance of the autoencoding models seems to improve in some cases, e.g., for the models trained on the IoT-23 data. Also, the AEGMM models were more stable during training. We discuss the possible reasons in Section 6.3.

- We observe quite low numbers for the test batch performance. That is not unexpected, as the low contamination rates and the total number of data points are not in the models' favor. The best results are shown by the NO_AGGR CICIDS and AGGR IoT-23 models. However, in these cases, the contamination

THRESHOLD BATCH

CTU-13	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	85	0.215	**0.640**	0.322	90	0.382	**0.764**	0.509	93	0.514	0.720	0.600
VAE	93	0.257	0.360	0.300	93	**0.450**	0.630	**0.525**	95	**0.588**	0.588	0.588
AEGMM	98	**0.726**	0.270	**0.394**	93	0.357	0.500	0.417	90	0.402	0.804	0.536
IF	90	0.250	0.500	0.333	90	0.337	0.674	0.449	90	0.407	**0.814**	0.543
MAH	97	0.373	0.224	0.280	93	0.400	0.560	0.467	93	0.540	0.756	**0.630**

CICIDS	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	97	0.28	0.168	0.21	97	0.823	0.494	0.617	95	0.73	**0.73**	**0.73**
VAE	97	**0.32**	0.192	0.24	93	0.543	**0.76**	**0.633**	95	0.704	0.704	0.704
AEGMM	80	0.216	0.862	**0.345**	97	0.824	0.488	0.613	97	0.803	0.482	0.602
IF	50	0.093	**0.928**	0.169	93	0.413	0.578	0.482	95	0.565	0.564	0.565
MAH	97	0.277	0.166	0.208	97	**0.84**	0.504	0.63	97	**0.807**	0.484	0.605

IoT-23	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	80	0.253	**1.000**	0.404	95	0.998	0.998	0.998	95	**1.000**	**1.000**	**1.000**
VAE	85	0.279	0.836	0.418	95	**1.000**	**1.000**	**1.000**	95	**1.000**	**1.000**	**1.000**
AEGMM	-	-	-	-	95	0.980	0.980	0.980	95	**1.000**	**1.000**	**1.000**
IF	70	0.247	**1.000**	0.397	90	0.488	0.976	0.651	90	0.501	**1.000**	0.667
MAH	88	**0.337**	0.808	**0.475**	95	0.996	0.996	0.996	95	0.994	0.994	0.994

TEST BATCH

CTU-13	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	85	**0.088**	**0.263**	**0.132**	90	0.020	0.706	0.038	93	0.012	0.686	0.023
VAE	93	0.018	0.026	0.021	93	**0.032**	0.677	**0.061**	95	**0.01**9	0.667	**0.036**
AEGMM	98	0.002	0.000	0.001	93	0.006	0.192	0.012	90	0.005	0.724	0.011
IF	90	0.027	0.075	0.040	90	0.017	0.692	0.032	90	0.007	0.710	0.013
MAH	97	0.055	0.043	0.048	93	0.007	**0.987**	0.015	93	0.000	**0.997**	0.001

CICIDS	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE	REC	F1	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	97	**0.886**	0.587	0.706	97	**0.125**	0.156	**0.139**	95	**0.095**	0.345	**0.149**
VAE	97	0.868	0.602	**0.711**	93	0.064	**0.341**	0.108	95	0.093	0.368	0.148
AEGMM	80	0.428	0.681	0.526	97	0.079	0.073	0.076	97	0.08	0.142	0.102
IF	50	0.244	**0.741**	0.367	93	0.045	0.296	0.078	95	0.069	0.393	0.117
MAH	97	0.746	0.225	0.345	97	0.027	0.244	0.048	97	0.035	**0.876**	0.066

IoT-23	**NO_AGGR**				**AGGR_M**				**AGGR_3M**			
	TH	PRE*	REC*	F1*	TH	PRE	REC	F1	TH	PRE	REC	F1
AE	80	**1.000**	0.000	0.001	95	0.564	0.569	0.567	95	**0.572**	0.571	0.572
VAE	85	0.392	0.272	0.321	95	0.564	0.569	0.566	95	0.571	0.571	0.571
AEGMM	-	-	-	-	95	**0.583**	0.618	0.600	95	0.570	0.565	0.567
IF	70	0.000	0.000	0.000	90	0.569	**0.961**	**0.715**	90	0.563	**0.995**	**0.719**
MAH	88	0.357	**0.555**	**0.434**	95	0.563	0.568	0.566	95	0.445	0.342	0.387

Table 22: Performance on the benchmark data split by scenario (TH: threshold, PRE: precision, REC: recall, F1: F_1-score). Best values highlighted in bold.

rate is relatively high (around 20%), which contributes to the higher precision and F_1-score values. The precision drops significantly in cases where the contamination rate is low (CTU-13 and AGGR CICIDS). However, we observe that the AGGR models achieve high recall values. This is a somewhat promising result, as the precision can potentially be improved later, e.g., by using a clever postprocessing procedure or another anomaly detector. As we explained above, for IoT-23 NO_AGGR we have inverted the definition for the positive class when computing F_1-score (marked with asterisks in the table).

In the next step of our analysis, we examine the performance of the models per attack scenario. Taking a different approach, we are going to analyze a few successful and unsuccessful attack detection cases. The complete performance per attack scenario results are provided in Appendix E.

Figure 20 shows the anomaly instance score plots of some successful attack detections. For the sake clarity, in a few plots below, we limit the highest anomaly scores with an upper value, i.e. score $= min(\text{true\_score}, \text{upper\_limit})$. To call an attack detection attempt successful, we require the model to achieve a high recall (the majority of the attack data points should be above the threshold) and a reasonable precision (the majority of the benign points should be below the threshold).

A discussion on the successful cases follows:

- *CTU-13*. Figure 20a shows the anomaly score plot of the VAE AGGR_M model on Scenario 9 (Neris). The model achieves good recall, but poor precision, failing to separate the benign and attack data points. We can also see that the initial stage of the attack (around 0:41:40), which could be that of a C%C communication, is not detected.

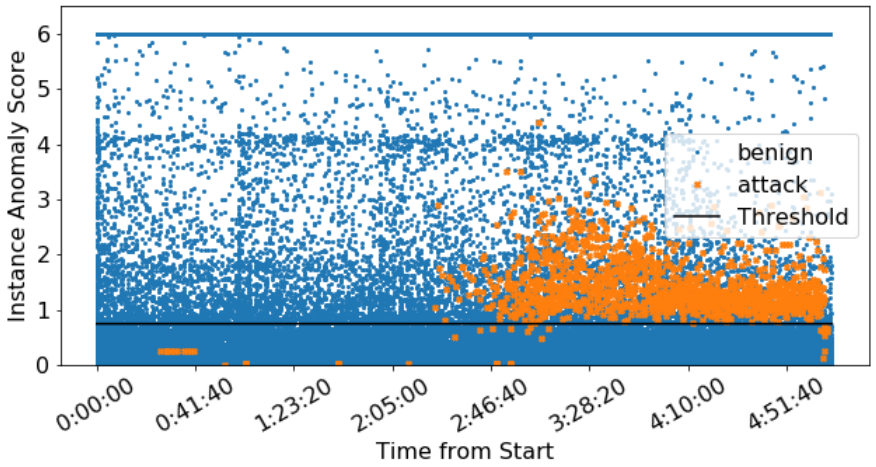
- *CICIDS 2017*. Figure 20b shows the anomaly score plot of the VAE AGGR_3M model on Scenario 8 (DoS). Similar to the case above, the model achieves good recall, but poor precision.

- *IoT-23*. Figure 20c shows the anomaly score plot of the VAE AGGR_3M model on Scenario 5 (Mirai 48). In this case, the model achieves excellent precision and recall, clearly separating the benign and attack data points.

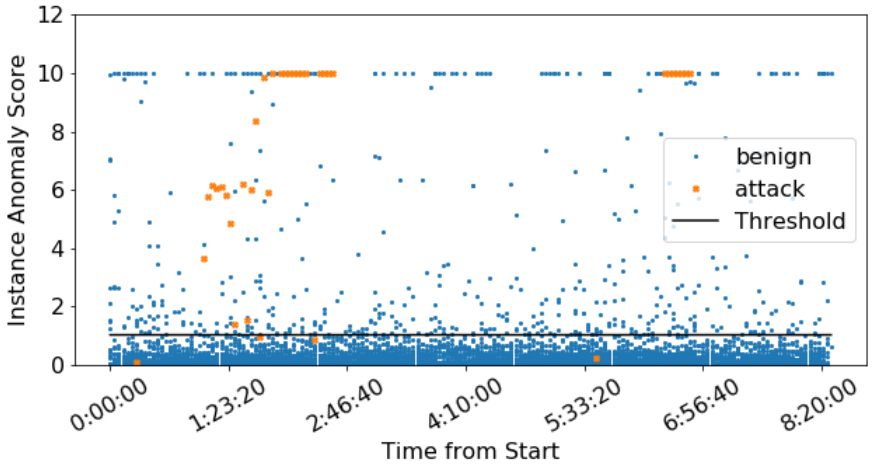
Figure 21 shows unsuccessful attack detection cases for each dataset. The reasons for failure are twofold: the models do a poor job at separating benign and attack data points, and the selected detection threshold does not work well in this particular case.

A discussion on the unsuccessful cases follows:

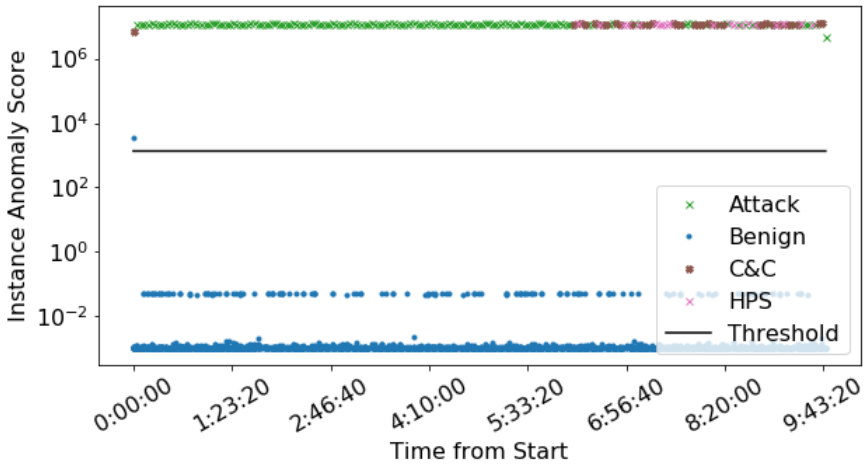
- *CTU-13*. Figure 21a shows the anomaly score plot of the MAH NO_AGGR model on Scenario 8 (Murlo). We can see that none of the attack data points get detected with the selected threshold. However, certain attack network flows, that also are periodic, are assigned a high anomaly score. If the threshold were selected a little lower (e.g. below 20), those attack data points would have gotten detected.



(a) VAE AGGR_M on Scenario 9 (Neris), CTU-13



(b) VAE AGGR_3M on Scenario 8 (DoS), CICIDS 2017



(c) VAE AGGR_3M on Scenario 5 (Mirai 48), IoT-23

Figure 20: Successful detection cases (high recall and precision).

- *CICIDS 2017.* Figure 21b shows the anomaly score plot of the AEGMM AGGR_3T model on Scenario 7 (Brute-Force). Here, the model fails to separate benign and attack data points.

- *IoT-23.* Figure 21c shows the anomaly score plot of the VAE AGGR_3T model on Scenario 8 (Mirai 34). Note that this is the same model as in the successful example from above. We can see that it manages to separate C&C data points from the benign data points, but they don't get flagged using the selected threshold. Nevertheless, we observe that the DDoS and Horizontal Port Scan (HPS) data points get detected in this example.

6.2.4 Aggregation of Anomaly Scores

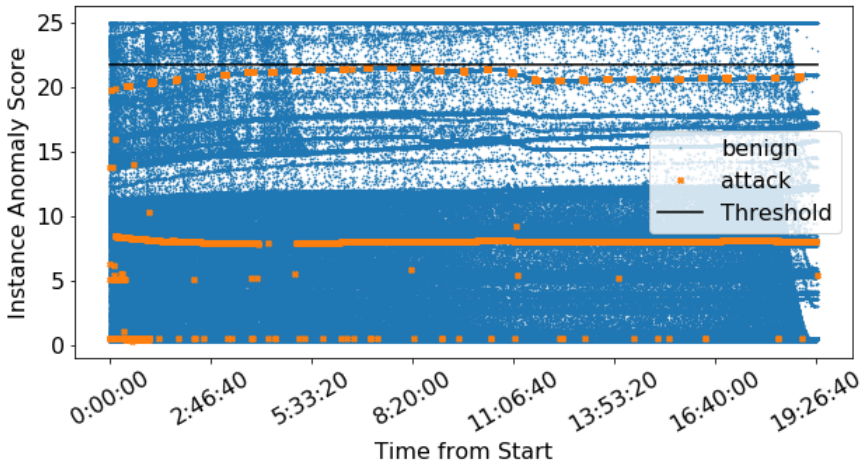
In this section, we showcase aggregating anomaly scores output by the NO_AGGR models. Our method for aggregating anomaly scores is explained in detail in Section 6.1.3.

Figure 22 shows a successful case of aggregating anomaly scores on the example of the NO_AGGR AEGMM model evaluated on Scenario 9 (Neris), CTU-13.

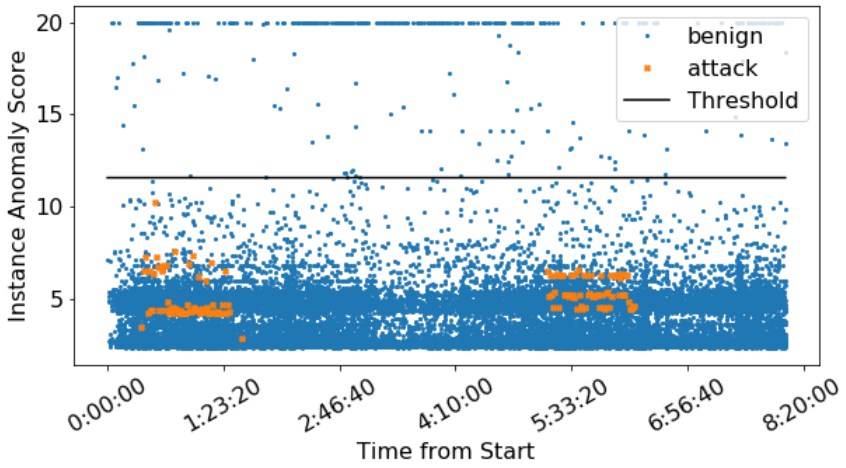
- Figure 22a shows the original instance score plot produced by the model. We can see from the plot that a lot of the true attack data points are missed.

- Figure 22b shows the result of aggregating the anomaly scores using our method. Here, we can see that we have "cleaned up" the plot, reducing the total number of data points while keeping the correct detections.

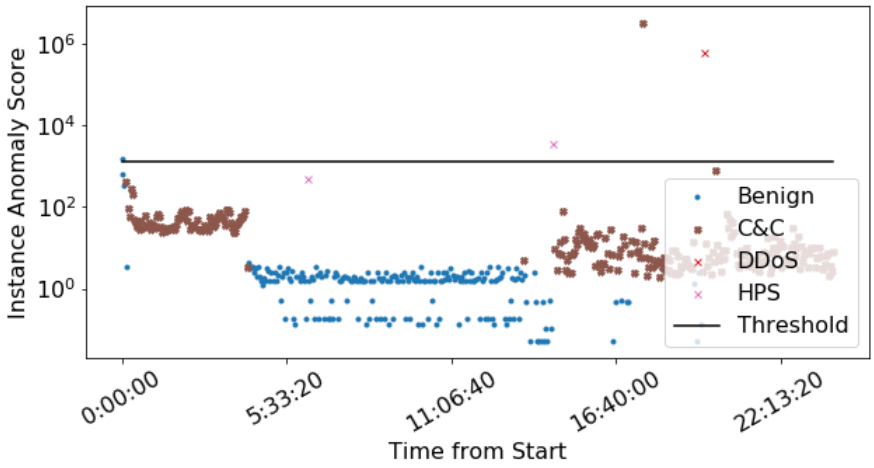
- For comparison, Figure 22c shows the anomaly instance score plot produced by the AGGR_3M AEGMM model on the same attack scenario. We can see that this model achieves good recall, but very low precision. Noticeably, the AGGR_3M AEGMM model correctly selects the beginning of the attack (around 0:41:40), while the NO_AGGR model misses it.



(a) MAH NO_AGGR on Scenario 8 (Murlo), CTU-13

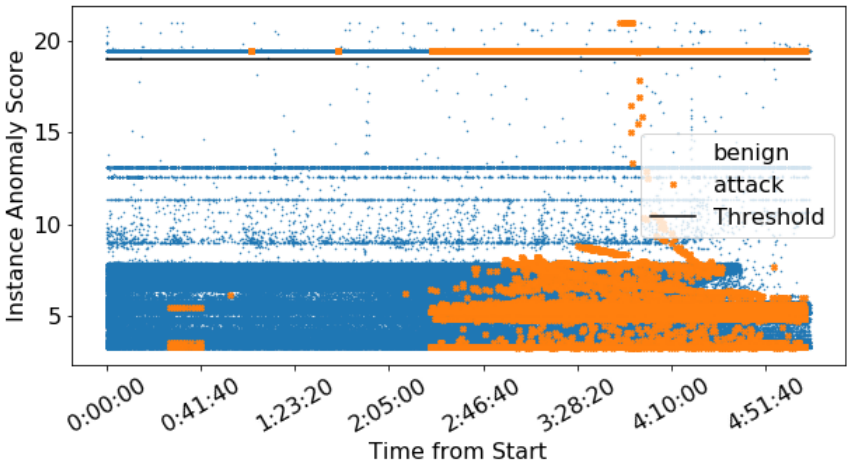


(b) AEGMM AGGR_3T on Scenario 7 (Brute-Force), CICIDS 2017

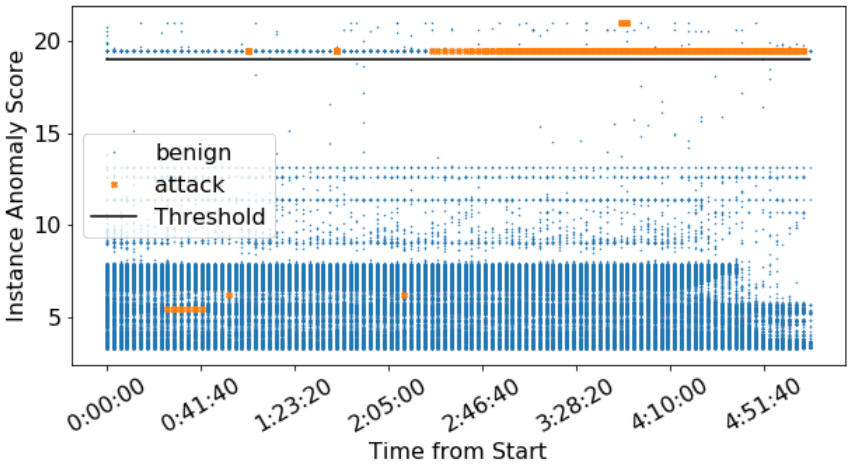


(c) VAE AGGR_3M on Scenario 8 (Mirai 34), IoT-23

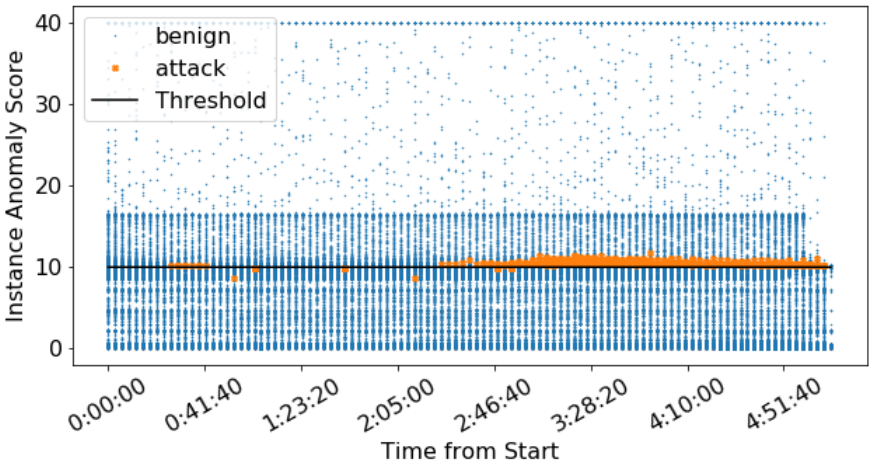
Figure 21: Unsuccessful detection cases (low recall and precision).



(a) AEGMM NO_AGGR on Scenario 9 (Neris), CTU-13



(b) Same as above after aggregating anomaly score over 3M



(c) AEGMM AGGR_3M model on the same scenario for reference

Figure 22: Aggregation of anomaly scores on Scenario 9 (Neris), CTU-13

6.3 Discussion

In the previous sections, we presented a methodology for proper training, testing, and comparing anomaly detection models for network intrusion detection. Here we summarize its main steps and the important findings.

The first step was to create appropriate benchmarks from the original data sources. We propose two approaches:

- *Split-at-random.* In this case, we split the data points at random and fix the contamination rates of the threshold and test batches. The performance numbers obtained on these benchmarks are not completely realistic, but allow us to conduct a meaningful comparison of the tested models.

  We obtain the value of $\chi_F^2 = 18.93$ for the Friedman statistic, which is higher than the critical value; thus, we reject the hypothesis that the models perform equally well. Figure 23 shows the obtained critical diagram, from which we can see that that VAE and AE generally outperform other models.

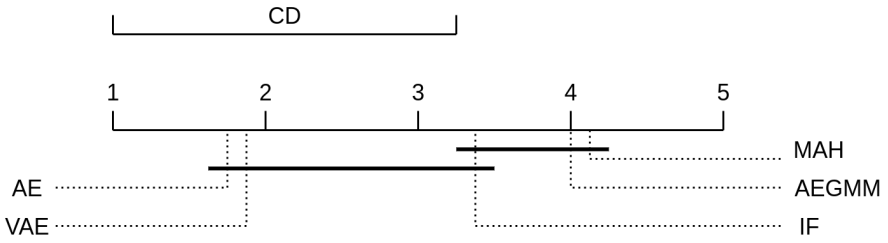


Figure 23: Critical difference diagram for the split-at-random benchmarks

- *Split-by-scenario* In this case, we split the data by attack scenario and keep the contamination rate of the test batch. These benchmarks are closer to a real-world environment, and allow us to test the algorithms on unseen attacks.

  The statistical tests give us similar results as in the previous case (see Figure 24). However, we observe that the models achieve very low precision; in other words, they produce a lot of false alarms.

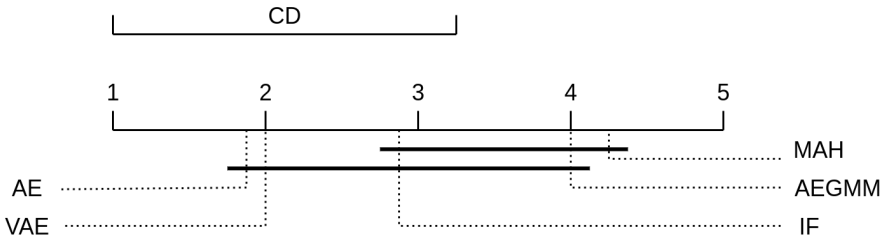


Figure 24: Critical difference diagram for the split-by-scenario benchmarks

We have also observed that our method for aggregating network flows provides a noticeable improvement in performance in many scenarios. The three-minute

aggregation window seems to perform better than the one-minute one, but the difference is not significant. We hypothesize that the optimal aggregation width can be found based on certain characteristics of the network traffic. We leave this question for future research.

We analyze the performance of the models for detecting different types of attacks. From the results on the split-at-random benchmarks, we observe that most of the attacks from all three datasets have good or moderate detectability. However, we find that the best performing models are different in each case, and none of the models excels at detecting every attack scenario.

From the results on the split-by-scenario benchmarks, we identify two reasons why a model that performs relatively well overall can have poor performance on certain attack scenarios. Firstly, we find that it may fail to separate the benign and attack data points efficiently in those attack scenarios. Secondly, we observe that even if the model separates well the benign and attack traffic, the selected threshold may not be optimal.

We propose a method for aggregating anomaly scores for NO_AGGR models, which shares a similar concept to network flow aggregation. We find that this method shows promising results on some attack scenarios, producing better scores than the corresponding AGGR models. We note that aggregated anomaly scores produced by a NO_AGGR models can easily be combined with the scores produced by NO_AGGR and AGGR models.

Lastly, we make a few observations on training the deep learning models and hyperparameter selection. From our experience, AE and VAE are quite straightforward to train, unlike AEGMM. We have encountered many numerical errors during AEGMM optimization and found that the outcome was inconsistent. We believe these issues might be caused by the following: certain properties of the data, implementation issues, or not optimal hyperparameter settings. Unfortunately, we have not been able to establish the root cause. Nevertheless, AEGMM seems to have good potential based on the successful cases, and it's suitability for network intrusion detection requires further investigation.

Another observation is that the deep learning models seem to obtain better threshold batch performance results on the split-by-scenario benchmarks, as compared to the split-at-random benchmarks. After conducting an ablation study, we find that this is caused by the way the data points are batched. In the split-by-scenario case, the data points are split into batches first, and then the batches are shuffled. Therefore, the batches contain contiguous bits of network traffic data (as the data points were sorted by timestamp initially). On the other hand, in the split-at-random case, the data points are shuffled first and then batched. We conclude that having "contiguous" batches is beneficial for optimization.

We have tried many different combinations of hyperparameters for each model (see Appendix C). Of course, we cannot guarantee that the selected settings are optimal, as there exists an infinite number of possible options, but we believe that they are adequate. However, it should be mentioned that we have not been to identify any trends in selecting hyperparameter values.

7 Conclusions

In this section, we give answers to the stated research questions based on the experimental findings. Next, we discuss the impact of the conducted research and provide recommendations. Finally, we identify the limitations of our study and propose questions for future research.

7.1 Answers to Research Questions

In this thesis, a selection of deep learning-based autoencoding models for anomaly detection, AE, VAE and AEGMM were compared against classical methods of IF and MAH. The comparison was conducted in a systematic way on three modern network intrusion datasets.

From the conducted experiments, we reject the hypothesis that the models perform equally well for network intrusion detection. We have shown that VAE and AE are robust and consistently outperform the IF and MAH. AEGMM performs inconsistently, and it is difficult to train. Our conclusion is that the deep-learning models show good potential for network intrusion detection and should be studied in more depth.

The second aim of this study was to investigate if network flow aggregation improves the performance. We have shown that it does, indeed, in most cases. It should be noted that other researches also support this claim [7, 17]. That said, we also have found that in a few scenarios (see Section 6.2.2 and Appendix D), the models trained on network flow features outperform the models trained on the aggregated features.

The third aim of this study was to analyze the performance of the methods on different types of attacks. We have found that the majority of the attacks has good or moderate detectability. Interestingly, we have observed that the best-performing methods are different for each attack. This may be interpreted in two ways: (1) different attack types exhibit certain characteristics that make them suitable for the application of a particular method; (2) the models "get lucky" during the optimization for detecting particular types of attacks.

Furthermore, we have found that one of the main challenges is the selection of an optimal threshold: although our method usually achieves a reasonably good performance on average, it may fail to detect certain attacks (see Section 6.2.3 and Appendix E). We conclude that a single anomaly threshold does not scale well to large datasets that contain heterogenous network traffic patterns.

Lastly, the results show that the methods produce a prohibitively large number of false alarms. Therefore, they cannot be employed *as is* in a real-world network, as it is not possible to go through all the alerts manually. That said, we observe the highest precision values on the network traffic from IoT devices. Therefore, we conclude that protecting IoT networks is one of the most promising application areas for our methods.

7.2 Recommendations

Computational complexity and reliability are as important in real-world environments as performance. We would recommend using AE and VAE as they showed to be the best-performing models within our experiments, and furthermore, they are straightforward to optimize, robust and allow for fast inference.

The study has identified that even the best performing methods tend to produce a large number of false alarms. Therefore, something must be done to improve the precision of the models. We can think of the following approaches:

- Ensemble several models to obtain better prediction performance.
- Cluster the traffic into more homogeneous groups and train a separate model for each group.
- Optimize the threshold selection process, considering making it adaptive for each network device or group.

7.3 Limitations

A large number of conducted experiments makes it difficult to validate all the findings and examine individual cases in greater depth. Furthermore, we have not been in close contact with the authors of the datasets, and thus, relied on the provided data and documentation. Consequently, it is possible that there are some mistakes in the results due to our misinterpretations, insufficient amount of data, and software and data bugs.

Secondly, due to the restrictions on computational resources and time, we have not been able to conduct enough experiments to ensure that the results are statistically accurate. We observe a lot of variance in the results which could affect some of the conclusions.

For the same reason, we have also not been able to explore the hyperparameter space in sufficient depth. While we believe that the selected settings are adequate, we cannot guarantee that they are optimal.

7.4 Future Research

Interesting potential directions for future research include:

- Evaluating other deep leaning-based algorithms, such as CNN- and RNN-based AEs, and adversarial AE.
- Studying the feature importance, for example using the methods described in [64].
- Investigating the performance of the deep learning models in the fully unsupervised mode.
- Finding better ways for threshold selection. For example, by selecting a separate threshold for each device based on its network profile.

References

[1] Ericsson. *Ericsson Mobility Report November 2019*. Nov. 2019. URL: https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf.

[2] *Security Aspects of 5G for Industrial Networks*. 2020. URL: https://www.5g-acia.org/publications/security-aspects-of-5g-for-industrial-networks/.

[3] Monowar Bhuyan, Dhruba K Bhattacharyya, and Jugal Kalita. *Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools*. Jan. 2017. ISBN: 978-3-319-65186-6. DOI: 10.1007/978-3-319-65188-0.

[4] A. Sperotto et al. "An Overview of IP Flow-Based Intrusion Detection". In: *IEEE Communications Surveys Tutorials* 12.3 (2010), pp. 343–356.

[5] R. Sommer and V. Paxson. "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". In: *2010 IEEE Symposium on Security and Privacy*. May 2010, pp. 305–316. DOI: 10.1109/SP.2010.25.

[6] Markus Ring et al. "A Survey of Network-based Intrusion Detection Data Sets". In: *CoRR* abs/1903.02460 (2019). arXiv: 1903.02460. URL: http://arxiv.org/abs/1903.02460.

[7] Sebastián García et al. "An Empirical Comparison of Botnet Detection Methods". In: *Computers & Security* 45 (Sept. 2014), pp. 100–123. DOI: 10.1016/j.cose.2014.05.011.

[8] Iman Sharafaldin, Arash Habibi Lashkari, and Ali Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: Jan. 2018, pp. 108–116. DOI: 10.5220/0006639801080116.

[9] Maria Jose Erquiaga Agustin Parmisano Sebastian Garcia. *Stratosphere Laboratory. A labeled dataset with malicious and benign IoT network traffic*. Jan. 2020. URL: https://www.stratosphereips.org/datasets-iot23.

[10] Gabriel Maciá-Fernández et al. "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs". In: *Computers & Security* 73 (Nov. 2017). DOI: 10.1016/j.cose.2017.11.004.

[11] Raghavendra Chalapathy and Sanjay Chawla. "Deep Learning for Anomaly Detection: A Survey". In: *CoRR* abs/1901.03407 (2019). arXiv: 1901.03407. URL: http://arxiv.org/abs/1901.03407.

[12] Vít Skvára, Tomás Pevný, and Václav Smídl. "Are generative deep models for novelty detection truly better?" In: *CoRR* abs/1807.05027 (2018). arXiv: 1807.05027. URL: http://arxiv.org/abs/1807.05027.

[13] Bo Zong et al. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=BJJLHbb0-.

[14] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[16] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: *IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA* 2 (July 2009). DOI: 10.1109/CISDA.2009.5356528.

[17] Quoc Phong Nguyen et al. "GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection". In: *CoRR* abs/1903.06661 (2019). arXiv: 1903.06661. URL: http://arxiv.org/abs/1903.06661.

[18] Christopher Kruegel and Giovanni Vigna. "Anomaly Detection of Web-Based Attacks". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*. CCS '03. Washington D.C., USA: Association for Computing Machinery, 2003, pp. 251–261. ISBN: 1581137389. DOI: 10.1145/948109.948144. URL: https://doi.org/10.1145/948109.948144.

[19] Matthew V. Mahoney and Philip K. Chan. "Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '02. Edmonton, Alberta, Canada: Association for Computing Machinery, 2002, pp. 376–385. ISBN: 158113567X. DOI: 10.1145/775047.775102. URL: https://doi.org/10.1145/775047.775102.

[20] Federico Simmross-Wattenberg et al. "Anomaly Detection in Network Traffic Based on Statistical Inference and alpha-Stable Modeling". In: *Dependable and Secure Computing, IEEE Transactions on* 8 (July 2011), pp. 494–509. DOI: 10.1109/TDSC.2011.14.

[21] Huy Anh Nguyen et al. "Network traffic anomalies detection and identification with flow monitoring". In: *2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN '08)*. May 2008, pp. 1–5. DOI: 10.1109/WOCN.2008.4542524.

[22] Eleazar Eskin et al. "A Geometric Framework for Unsupervised Anomaly Detection". In: *Applications of Data Mining in Computer Security*. Ed. by Daniel Barbará and Sushil Jajodia. Boston, MA: Springer US, 2002, pp. 77–101. ISBN: 978-1-4615-0953-0. DOI: 10.1007/978-1-4615-0953-0_4. URL: https://doi.org/10.1007/978-1-4615-0953-0_4.

[23] Haakon Ringberg et al. "Sensitivity of PCA for traffic anomaly detection". In: vol. 35. June 2007, pp. 109–120. DOI: 10.1145/1254882.1254895.

[24] Daniela Brauckhoff, Kave Salamatian, and Martin May. "Applying PCA for Traffic Anomaly Detection: Problems and Solutions". In: May 2009, pp. 2866–2870. DOI: 10.1109/INFCOM.2009.5062248.

[25] Yoshiki Kanda et al. "ADMIRE: Anomaly detection method using entropy-based PCA with three-step sketches". In: *Computer Communications* 36.5 (2013), pp. 575–588. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2012.12.002. URL: http://www.sciencedirect.com/science/article/pii/S0140366412003994.

[26] Nour Moustafa and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)". In: Nov. 2015. DOI: 10.1109/MilCIS.2015.7348942.

[27] Leyla Bilge et al. "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis". In: Dec. 2012, pp. 129–138. DOI: 10.1145/2420950.2420969.

[28] Donghwoon Kwon et al. "An Empirical Study on Network Anomaly Detection Using Convolutional Neural Networks". In: July 2018, pp. 1595–1598. DOI: 10.1109/ICDCS.2018.00178.

[29] Pablo Torres et al. "An Analysis of Recurrent Neural Networks for Botnet Behavior Detection". In: June 2016. DOI: 10.1109/ARGENCON.2016.7585247.

[30] N. Shone et al. "A Deep Learning Approach to Network Intrusion Detection". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.1 (2018), pp. 41–50.

[31] Pierre Baldi. "Autoencoders, Unsupervised Learning, and Deep Architectures". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon et al. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, Feb. 2012, pp. 37–49. URL: http://proceedings.mlr.press/v27/baldi12a.html.

[32] Yang Yu, Jun Long, and Zhiping Cai. "Network Intrusion Detection through Stacking Dilated Convolutional Autoencoders". In: *Security and Communication Networks* 2017 (Nov. 2017), pp. 1–10. DOI: 10.1155/2017/4184196.

[33] Yisroel Mirsky et al. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection". In: Jan. 2018. DOI: 10.14722/ndss.2018.23211.

[34] Jinwon An and Sungzoon Cho. "Variational Autoencoder based Anomaly Detection using Reconstruction Probability". In: 2015.

[35] Haowen Xu et al. "Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications". In: *CoRR* abs/1802.03903 (2018). arXiv: 1802.03903. URL: http://arxiv.org/abs/1802.03903.

[36] Manuel Lopez-Martin et al. "Conditional Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion Detection in IoT". In: *Sensors* 17 (Aug. 2017), p. 1967. DOI: 10.3390/s17091967.

[37] Filipe Falcão et al. "Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection". In: Apr. 2019, pp. 318–327. DOI: 10.1145/3297280.3297314.

[38] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. 6th. Pearson, 2012. ISBN: 0132856204.

[39] Achiv Chauha and Palak Jain. *TCP/IP Model*. 2020. URL: https://www.geeksforgeeks.org/tcp-ip-model/.

[40] Science Commerce. "A "Kill Chain" analysis of the 2013 target data breach". In: Jan. 2014, pp. 41–60.

[41] *Cloudflare Learning Center*. 2020. URL: https://www.cloudflare.com/learning/.

[42] *Mirai (malware)*. 2020. URL: https://en.wikipedia.org/wiki/Mirai_(malware).

[43] *ENISA Threat Landscape Report 2018: 15 Top Cyberthreats and Trends*. European Union Agency For Network and Information Security, 2019. ISBN: 9789292042868. URL: https://books.google.fi/books?id=oUHDwQEACAAJ.

[44] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[45] William W. Cohen. "Fast Effective Rule Induction". In: *Machine Learning Proceedings 1995*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann, 1995, pp. 115–123. ISBN: 978-1-55860-377-6. DOI: https://doi.org/10.1016/B978-1-55860-377-6.50023-2. URL: http://www.sciencedirect.com/science/article/pii/B9781558603776500232.

[46] Guilherme Campos et al. "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". In: *Data Mining and Knowledge Discovery* 30 (Jan. 2016). DOI: 10.1007/s10618-015-0444-8.

[47] Jon Louis Bentley. "K-d Trees for Semidynamic Point Sets". In: *Proceedings of the Sixth Annual Symposium on Computational Geometry*. SCG '90. Berkley, California, USA: Association for Computing Machinery, 1990, pp. 187–197. ISBN: 0897913620. DOI: 10.1145/98524.98564. URL: https://doi.org/10.1145/98524.98564.

[48] Anant Ram et al. "A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases". In: *International Journal of Computer Applications* 3 (June 2010). DOI: 10.5120/739-1038.

[49] François Chollet. *Building Autoencoders in Keras*. May 2016. URL: https://blog.keras.io/building-autoencoders-in-keras.html.

[50] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. 2014. arXiv: 1401.4082 [stat.ML].

[51] F. T. Liu, K. M. Ting, and Z. Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422.

[52] P. C. Mahalanobis. “On the generalized distance in statistics”. In: 1936.

[53] Arnaud Van Looveren et al. *Alibi-Detect: Algorithms for outlier and adversarial instance detection, concept drift and metrics.* Version 0.3.0. Feb. 17, 2020. URL: https://github.com/SeldonIO/alibi-detect.

[54] Calvin Ko, Manfred Ruschitzka, and Karl Levitt. “Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-Based Approach.” In: Jan. 1997, pp. 175–187.

[55] Ayyoob Hamza et al. “Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles (Technical Report)”. In: *CoRR* abs/1804.04358 (2018). arXiv: 1804.04358. URL: http://arxiv.org/abs/1804.04358.

[56] B Claise, Brian Trammell, and P Aitken. “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information”. In: (Sept. 2013).

[57] Muhammad Umer, Muhammad Sher, and Yaxin Bi. “Flow-based intrusion detection: Techniques and challenges”. In: *Computers & Security* 70 (June 2017). DOI: 10.1016/j.cose.2017.05.009.

[58] M. Ring et al. “IP2Vec: Learning Similarities Between IP Addresses”. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2017, pp. 657–666.

[59] Sebastián García et al. *The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic.* 2013. URL: https://www.stratosphereips.org/datasets-ctu13.

[60] Richard Lippmann et al. “The 1999 DARPA Off-Line Intrusion Detection Evaluation”. In: *Comput. Netw.* 34.4 (Oct. 2000), pp. 579–595. ISSN: 1389-1286. DOI: 10.1016/S1389-1286(00)00139-0. URL: https://doi.org/10.1016/S1389-1286(00)00139-0.

[61] S. Stolfo. *KDD Cup 1999 Data.* 1999. URL: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[62] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[63] Janez Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1–30. ISSN: 1532-4435.

[64] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers & Electrical Engineering* 40.1 (2014). 40th-year commemorative issue, pp. 16–28. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2013.11.024. URL: http://www.sciencedirect.com/science/article/pii/S0045790613003066.

[65] Mohammad Saiful Islam Mamun Arash Habibi Lashkari Gerard Draper-Gil and Ali A. Ghorbani. *Network Traffic Flow Analyzer.* 2016. URL: http://www.netflowmeter.ca/netflowmeter.html.

[66] *Zeek Manual*. 2020. URL: https://docs.zeek.org/en/current/index.html.

A Network flow and Aggregated Features

This section presents the features used for training the models. For each dataset, we specify both the network flow features and the aggregated features.

We use the following abbreviations for feature types: num = numerical, cat = categorical, bin = binary. We standardize the numerical features by removing the mean and scaling to unit variance (standard scaling). We encode the categorical features using OHE. We encode the binary features with a single binary column.

#	**Feature**	**Type**	**Description**
1	*proto*	cat	Flow protocol
2	*dur*	num	Flow total duration
3	*fwd_dir*	bin	Data sent in forward direction
4	*bwd_dir*	bin	Data sent in backward direction
5	*fwd_fin_flag*	bin	TCP sent ACK flag
6	*fwd_syn_flag*	bin	TCP sent SYN flag
7	*fwd_rst_flag*	bin	TCP sent RST flag
8	*fwd_psh_flag*	bin	TCP sent PSH flag
9	*fwd_ack_flag*	bin	TCP sent ACK flag
10	*fwd_urg_flag*	bin	TCP sent UGR flag
11	*fwd_cwe_flag*	bin	TCP sent CWE flag
12	*fwd_ece_flag*	bin	TCP sent ECE flag
13	*bwd_fin_flag*	bin	TCP received ECE flag
14	*bwd_syn_flag*	bin	TCP received SYN flag
15	*bwd_rst_flag*	bin	TCP received RST flag
16	*bwd_psh_flag*	bin	TCP received PSH flag
17	*bwd_ack_flag*	bin	TCP received ACK flag
18	*bwd_urg_flag*	bin	TCP received URG flag
19	*bwd_cwe_flag*	bin	TCP received CWE flag
20	*bwd_ece_flag*	bin	TCP received ECE flag
21	*src_tos*	bin	Source Type of Service (TOS) is non-zero
22	*dst_tos*	bin	Dest. Type of Service (TOS) is non-zero
23	*tot_pkts*	num	Total num. of pkts. exchanged
24	*tot_byts*	num	Total num. of bytes exchanged
25	*src_byts*	num	Num. of bytes sent by the source

Table A1: CTU-13 network flow features

#	Feature	Type	Description
1	*total_cnt*	num	Total num. of flows in a window
2	*dur_mean*	num	Mean flow duration
3	*dur_min*	num	Min. flow duration
4	*dur_max*	num	Max. flow duration
5	*dur_std*	num	SD of flow duration
6	*dst_ip_entropy*	num	Entropy of destination IPs
7	*dst_ip_nuniq*	num	Num. of unique destination IPs
8	*dst_port_entropy*	num	Entropy of destination IPs
9	*dst_port_nuniq*	num	Num. of unique destination IPs
10	*src_port_entropy*	num	Entropy of source IPs
11	*src_port_nuniq*	num	Num. of unique source IPs
12	*proto_entropy*	num	Entropy of protocols
13	*proto_nuniq*	num	Num. of unique protocols
14	*fwd_flag_entropy*	num	Entropy of sent TCP flags
15	*fwd_flag_nuniq*	num	Num. of unique sent TCP flags
16	*bwd_flag_entropy*	num	Entropy of received TCP flags
17	*bwd_flag_nuniq*	num	Num. of unique received TCP flags
18	*tot_pkts_mean*	num	Mean total num. of pkts.
19	*tot_pkts_min*	num	Min. total num. of pkts.
20	*tot_pkts_max*	num	Max. total num. of pkts.
21	*tot_pkts_std*	num	SD of total num. of pkts.
22	*tot_pkts_median*	num	Median total num. of pkts.
23	*tot_byts_mean*	num	Mean total num. or bytes
24	*tot_byts_min*	num	Min. total num. or bytes
25	*tot_byts_max*	num	Max. total num. or bytes
26	*tot_byts_std*	num	SD of total numbenumr bytes
27	*tot_byts_median*	num	Median of total num. or bytes
28	*src_byts_mean*	num	Mean num. of bytes sent by the source
29	*src_byts_min*	num	Min. num. of bytes sent by the source
30	*src_byts_max*	num	Max. num. of bytes sent by the source
31	*src_byts_std*	num	SD of num. of bytes sent by the source
32	*src_byts_median*	num	Median num. of bytes sent by the source

Table A2: CTU-13 aggregated features

#	Feature	Type	Description
1	*protocol*	cat	Flow protocol
2	*dur*	num	Flow duration
3	*tot_fwd_pkt*	num	Total num. of pkts. in fwd. dir.
4	*tot_bwd_pkts*	num	Total num. of pkts. in bwd. dir.
5	*tot_len_fwd_pkts*	num	Total size of pkts. in fwd. dir.
6	*tot_len_bwd_pkts*	num	Total size of pkts. in bwd. dir.
7	*fwd_pkt_len_max*	num	Max. packet size in fwd. dir.
8	*fwd_pkt_len_min*	num	Min. packet size in fwd. dir.
9	*fwd_pkt_len_mean*	num	Min. packet size in fwd. dir.
10	*fwd_pkt_len_std*	num	SD of packet size in fwd. dir.
11	*bwd_pkt_len_max*	num	Max. packet size in bwd. dir.
12	*bwd_pkt_len_min*	num	Min. packet size in bwd. dir.
13	*bwd_pkt_len_mean*	num	Mean packet size in bwd. dir.
14	*bwd_pkt_len_std*	num	SD of packet size in bwd. dir.
15	*flow_byts/s*	num	Num. of flow bytes per sec.
16	*flow_pkts/s*	num	Num. of packet bytes per sec.
17	*flow_iat_mean*	num	Mean time btw. two pkts.
18	*flow_iat_std*	num	SD of time btw. two pkts.
19	*flow_iat_max*	num	Max. time btw. two pkts.
20	*flow_iat_min*	num	Min. time btw. two pkts.
21	*fwd_iat_tot*	num	Total time btw. two pkts. in fwd. dir.
22	*fwd_iat_mean*	num	Mean time btw. two pkts. in fwd. dir.
23	*fwd_iat_std*	num	SD of time btw. two pkts. in fwd. dir.
24	*fwd_iat_max*	num	Max. time btw. two pkts. in fwd. dir.
25	*fwd_iat_min*	num	Min. time btw. two pkts. in fwd. dir.
26	*bwd_iat_tot*	num	Total time btw. two pkts. in bwd. dir.
27	*bwd_iat_mean*	num	Mean time btw. two pkts. in bwd. dir.
28	*bwd_iat_std*	num	SD of time btw. two pkts. in bwd. dir.
29	*bwd_iat_max*	num	Max. time btw. two pkts. in bwd. dir.
30	*bwd_iat_min*	num	Min. time btw. two pkts. in bwd. dir.
31	*fwd_psh_flags*	num	Num. of PSH flags in fwd. dir. (TCP)
32	*bwd_psh_flags*	num	Num. of PSH flags in bwd. dir. (TCP)
33	*fwd_urg_flags*	num	Num. of UGR flags in fwd. dir. (TCP)
34	*bwd_urg_flags*	num	Num. of UGR flags in bwd. dir. (TCP)
35	*fwd_header_len*	num	Total bytes used for headers in fwd. dir.
36	*bwd_header_len*	num	Total bytes used for headers in bwd. dir.
37	*fwd_pkts/s*	num	Num. of fwd. pkts. per sec.
38	*bwd_pkts/s*	num	Num. of bwd. pkts. per sec.
39	*pkt_len_min*	num	Min. packet size
40	*pkt_len_max*	num	Max. packet size
41	*pkt_len_mean*	num	Mean packet size

CICIDS 2017 network flow features

Continued on next page

42	*pkt_len_std*	num	SD of packet size
43	*pkt_len_var*	num	Variance of packet size
44	*fin_flag_cnt*	num	Num. of pkts. with FIN flag
45	*syn_flag_cnt*	num	Num. of pkts. with SYN flag
46	*rst_flag_cnt*	num	Num. of pkts. with RST flag
47	*psh_flag_cnt*	num	Num. of pkts. with PSH flag
48	*ack_flag_cnt*	num	Num. of pkts. with ACK flag
49	*urg_flag_cnt*	num	Num. of pkts. with URG flag
50	*cwe_flag_cnt*	num	Num. of pkts. with CWE flag
51	*ece_flag_cnt*	num	Num. of pkts. with ECE flag
52	*down/up_ratio*	num	Download and upload ratio
53	*fwd_seg_size_avg*	num	Avg. size observed in fwd. dir.
54	*bwd_seg_size_avg*	num	Avg. size observed in bwd. dir.
55	*fwd_byts/blk_avg*	num	Avg. num. of bytes bulk rate in fwd. dir.
56	*fwd_pkts/blk_avg*	num	Avg. num. of pkts. bulk rate in fwd. dir.
57	*fwd_blk_rate_avg*	num	Avg. num. bulk rate in fwd. dir.
58	*bwd_byts/blk_avg*	num	Avg. num. of bytes bulk rate in bwd. dir.
59	*bwd_pkts/blk_avg*	num	Avg. num. of pkts. bulk rate in bwd. dir.
60	*bwd_blk_rate_avg*	num	Avg. bulk rate in the bwd. dir.
61	*subflow_fwd_pkts*	num	Avg. num. of pkts. in a subflow in fwd. dir.
62	*subflow_fwd_byts*	num	Avg. num. of bytes in a subflow in fwd. dir.
63	*subflow_bwd_pkts*	num	Avg. num. of pkts. in a subflow in bwd. dir.
64	*subflow_bwd_byts*	num	Avg. num. of bytes in a subflow in bwd. dir.
65	*init_fwd_win_byts*	num	Num. of bytes sent in initial window in fwd. dir.
66	*init_bwd_win_byts*	num	Num. of bytes sent in initial window in bwd. dir.
67	*fwd_act_data_pkts*	num	Count of pkts. with at least 1 byte of TCP data payload in the fwd. dir.
68	*fwd_seg_size_min*	num	Min. segment size observed in fwd. dir.
69	*active_mean*	num	Mean time flow was active before idle
70	*active_std*	num	SD of time flow was active before idle
71	*active_max*	num	Max. time flow was active before idle
72	*active_min*	num	Min. time flow was active before idle
73	*idle_mean*	num	Mean time flow was idle before active
74	*idle_std*	num	SD of time flow was idle before active
75	*idle_max*	num	Max. time flow was idle before active
76	*idle_min*	num	Mix. time flow was idle before active

Table A3: CICIDS 2017 network flow features, adapted from [65]

#	Feature	Type	Description
1	*total_cnt*	num	Total num. of flows in a window
2	*dur_mean*	num	Mean flow duration
3	*dur_min*	num	Min. flow duration
4	*dur_max*	num	Max. flow duration
5	*dur_std*	num	SD of flow duration
6	*dst_ip_entropy*	num	Entropy of destination IPs
7	*dst_ip_nuniq*	num	Num. of unique destination IPs
8	*dst_port_entropy*	num	Entropy of destination IPs
9	*dst_port_nuniq*	num	Num. of unique destination IPs
10	*src_port_entropy*	num	Entropy of source IPs
11	*src_port_nuniq*	num	Num. of unique source IPs
12	*proto_entropy*	num	Entropy of protocols
13	*proto_nuniq*	num	Num. of unique protocols
14	*flag_entropy*	num	Entropy of exchanged TCP flags
15	*flag_nuniq*	num	Num. of unique exchanged TCP flags
16	*tot_fwd_pkts_mean*	num	Mean total num. of pkts. in fwd. dir.
17	*tot_fwd_pkts_min*	num	Min. total num. of pkts. in fwd. dir.
18	*tot_fwd_pkts_max*	num	Max. total num. of pkts. in fwd. dir.
19	*tot_fwd_pkts_std*	num	SD of total num. of pkts. in fwd. dir.
20	*tot_fwd_pkts_median*	num	Median total num. of pkts. in fwd. dir.
21	*tot_bwd_pkts_mean*	num	Mean total num. of pkts. in bwd. dir.
22	*tot_bwd_pkts_min*	num	Min. total num. of pkts. in bwd. dir.
23	*tot_bwd_pkts_max*	num	Max. total num. of pkts. in bwd. dir.
24	*tot_bwd_pkts_std*	num	SD of total num. of pkts. in bwd. dir.
25	*tot_bwd_pkts_median*	num	Median total num. of pkts. in bwd. dir.
26	*tot_len_fwd_pkts_mean*	num	Mean size of pkts. in fwd. dir.
27	*tot_len_fwd_pkts_min*	num	Min. size of pkts. in fwd. dir.
28	*tot_len_fwd_pkts_max*	num	Max. size of pkts. in fwd. dir.
29	*tot_len_fwd_pkts_std*	num	SD of size of pkts. in fwd. dir.
30	*tot_len_fwd_pkts_median*	num	Median size of pkts. in fwd. dir.
31	*tot_len_bwd_pkts_mean*	num	Mean size of pkts. in bwd. dir.
32	*tot_len_bwd_pkts_min*	num	Min. size of pkts. in bwd. dir.
33	*tot_len_bwd_pkts_max*	num	Max. size of pkts. in bwd. dir.
34	*tot_len_bwd_pkts_std*	num	SD of size of pkts. in bwd. dir.
35	*tot_len_bwd_pkts_median*	num	Median size of pkts. in bwd. dir.
36	*flow_byts/s_mean*	num	Mean num. of flow bytes per sec.
37	*flow_byts/s_min*	num	Min. num. of flow bytes per sec.
38	*flow_byts/s_max*	num	Max. num. of flow bytes per sec.
39	*flow_byts/s_std*	num	SD of num. of flow bytes per sec.

CICIDS 2017 aggregated features

Continued on next page

40	*flow_byts/s_median*	num	Median num. of flow bytes per sec.
41	*flow_pkts/s_mean*	num	Mean num. of flow pkts. per sec.
42	*flow_pkts/s_min*	num	Min. num. of flow pkts. per sec.
43	*flow_pkts/s_max*	num	Max. num. of flow pkts. per sec.
44	*flow_pkts/s_std*	num	SD of num. of flow pkts. per sec.
45	*flow_pkts/s_median*	num	Median num. of flow pkts. per sec.

Table A4: CICIDS 2017 aggregated features

#	Feature	Type	Description
1	*proto*	cat	Transport level protocol
2	*service*	cat	Dynamically detected application protocol, if any
3	*dur*	num	Flow total duration
4	*orig_bytes*	num	Originator payload bytes
5	*resp_bytes*	num	Responder payload bytes
6	*conn_state*	cat	Connection state
7	*missed_bytes*	num	Num. of missing bytes
8	*history_empty*	bin	History is empty
9	*history_dir_flipped*	bin	Connection direction was flipped by flow exporter
10-21	*orig_history_[l]_cnt*	num	Count of each originator history letter l
22-33	*resp_history_[l]_cnt*	num	Count of each responder history letter l
34	*orig_pkts*	num	Num. of originator packets
35	*orig_ip_bytes*	num	Num. of originator IP bytes
36	*resp_pkts*	num	Num. of responder packets
37	*resp_ip_bytes*	num	Num. of responder IP bytes

Table A5: IoT-23 network flow features, adapted from [66]

#	Feature	Type	Description
1	*total_cnt*	num	Total num. of flows in a window
2	*dur_mean*	num	Mean flow duration
3	*dur_min*	num	Min. flow duration
4	*dur_max*	num	Max. flow duration
5	*dur_std*	num	SD of flow duration
6	*dur_median*	num	Median flow duration

IoT-23 aggregated features
Continued on next page

7	*dst_ip_entropy*	num	Entropy of destination IPs
8	*dst_ip_nuniq*	num	Num. of unique destination IPs
9	*dst_port_entropy*	num	Entropy of destination ports
10	*dst_port_nuniq*	num	Num. of unique destination ports
11	*src_port_entropy*	num	Entropy of source ports
12	*src_port_nuniq*	num	Num. of unique source ports
13	*proto_entropy*	num	Entropy of transport protocols
14	*proto_nuniq*	num	Num. of unique transport protocols
15	*service_entropy*	num	Entropy of detected application protocols
16	*service_nuniq*	num	Num. of unique detected application protocols
17	*orig_hist_entropy*	num	Entropy of originator history letters
18	*orig_hist_nuniq*	num	Num. of unique originator history letters
19	*resp_hist_entropy*	num	Entropy of responder history letters
20	*resp_hist_nuniq*	num	Num. of unique responder history letters
21	*orig_bytes_mean*	num	Mean originator payload bytes
22	*orig_bytes_min*	num	Min. originator payload bytes
23	*orig_bytes_max*	num	Max. originator payload bytes
24	*orig_bytes_std*	num	SD of originator payload bytes
25	*orig_bytes_median*	num	Median originator payload bytes
26	*resp_bytes_mean*	num	Mean responder payload bytes
27	*resp_bytes_min*	num	Min. responder payload bytes
28	*resp_bytes_max*	num	Max. responder payload bytes
29	*resp_bytes_std*	num	SD of responder payload bytes
30	*resp_bytes_median*	num	Median responder payload bytes
31	*conn_state_entropy*	num	Entropy of connection states
32	*conn_state_nuniq*	num	Num. of unique connection states
33	*missed_bytes_mean*	num	Mean missing bytes
34	*missed_bytes_min*	num	Min. missing bytes
35	*missed_bytes_max*	num	Max. missing bytes
36	*missed_bytes_std*	num	SD of missing bytes
37	*missed_bytes_median*	num	Median missing bytes
38	*orig_pkts_mean*	num	Mean originator packets
39	*orig_pkts_min*	num	Min. originator packets
40	*orig_pkts_max*	num	Max. originator packets
41	*orig_pkts_std*	num	SD of originator packets
42	*orig_pkts_median*	num	Median originator packets
43	*orig_ip_bytes_mean*	num	Mean originator IP bytes
44	*orig_ip_bytes_min*	num	Min. originator IP bytes
45	*orig_ip_bytes_max*	num	Max. originator IP bytes
46	*orig_ip_bytes_std*	num	SD of originator IP bytes
47	*orig_ip_bytes_median*	num	Median originator IP bytes
48	*resp_pkts_mean*	num	Mean responder packets

IoT-23 aggregated features

Continued on next page

49	*resp_pkts_min*	num	Min. responder packets
50	*resp_pkts_max*	num	Max. responder packets
51	*resp_pkts_std*	num	SD of responder packets
52	*resp_pkts_median*	num	Median responder packets
53	*resp_ip_bytes_mean*	num	Mean responder IP bytes
54	*resp_ip_bytes_min*	num	Min. responder IP bytes
55	*resp_ip_bytes_max*	num	Max. responder IP bytes
56	*resp_ip_bytes_std*	num	SD of responder IP bytes
57	*resp_ip_bytes_median*	num	Median responder IP bytes

Table A6: IoT-23 aggregated features

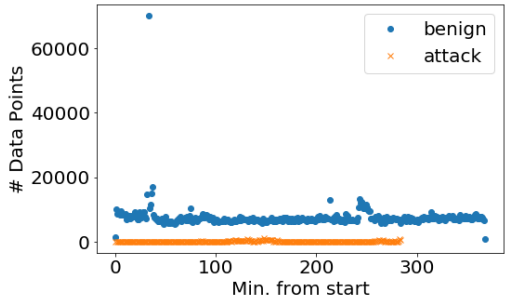
B Attack Scenarios

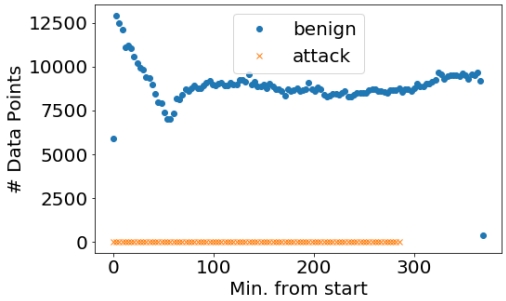
This section explores the attack scenarios for each dataset, as identified in Section 5. For each attack scenario, we give the number of benign and attack data points (contamination rate is specified in parenthesis) and plot the number of data points per minute. We provide the information for two versions of the data: network flows (NO_AGGR) and aggregations of flows over a three-minute window (AGGR_3M).

CTU-13

1 Neris

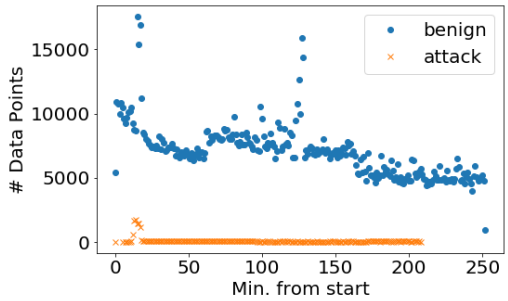
NO_AGGR	AGGR_3M
# Benign: 2783648	# Benign: 1102475
# Attack: 40961 (1.45%)	# Attack: 96 (0.01%)

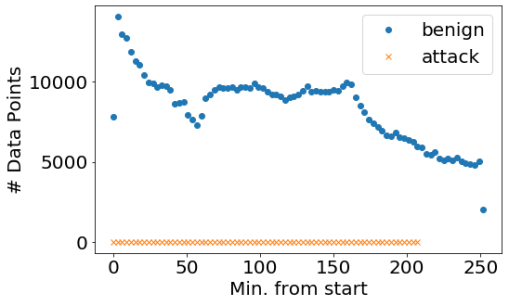




2 Neris

NO_AGGR	AGGR_3M
# Benign: 1787163	# Benign: 709963
# Attack: 20941 (1.16%)	# Attack: 70 (0.01%)



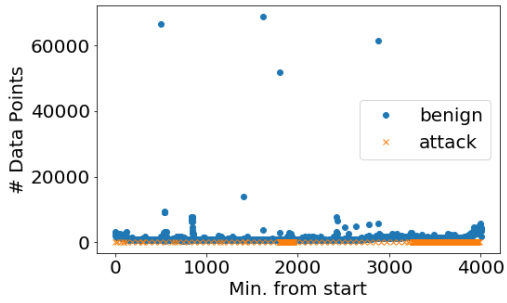


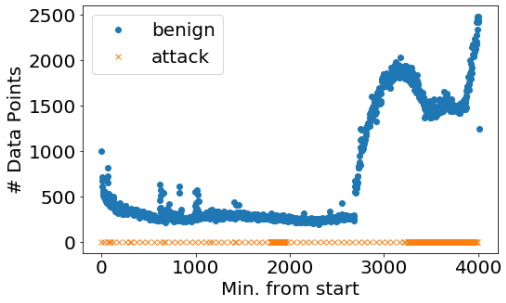
CTU-13 attack scenarios

Continued on next page

3 Rbot

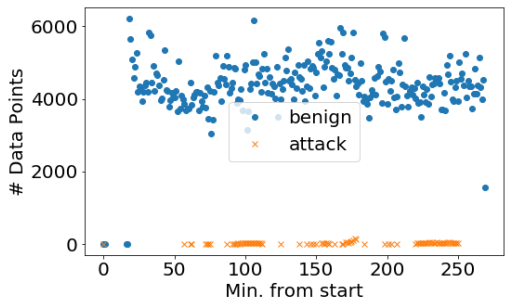
NO_AGGR	AGGR_3M
# Benign: 4683587	# Benign: 966874
# Attack: 26822 (0.57%)	# Attack: 362 (0.04%)

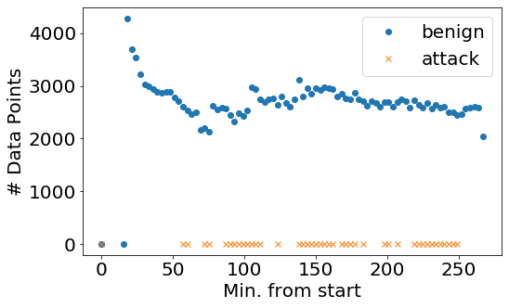




4 Rbot

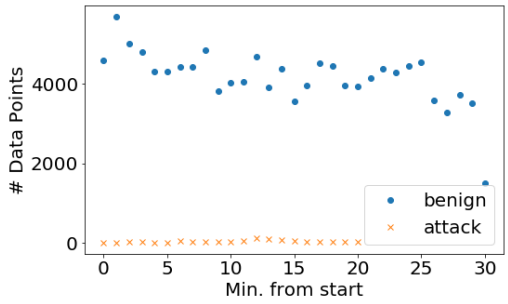
NO_AGGR	AGGR_3M
# Benign: 1118482	# Benign: 228898
# Attack: 2580 (0.23%)	# Attack: 43 (0.02%)

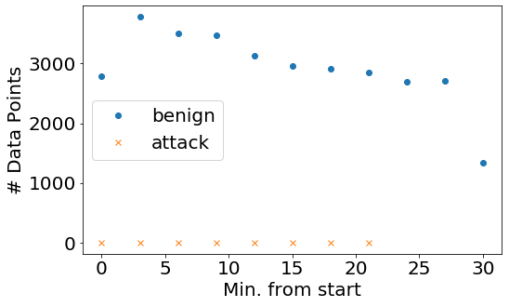




5 Virut

NO_AGGR	AGGR_3M
# Benign: 128929	# Benign: 32128
# Attack: 901 (0.69%)	# Attack: 8 (0.02%)

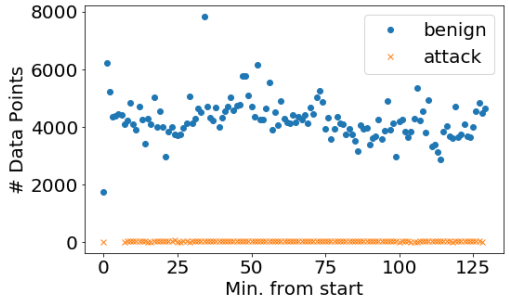


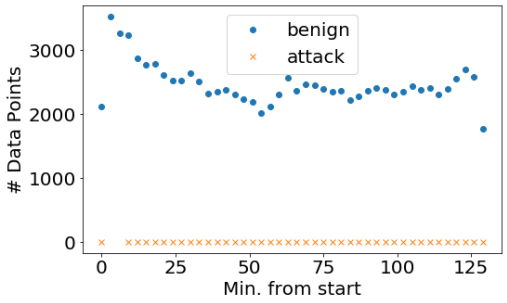


CTU-13 attack scenarios
Continued on next page

6 Menti

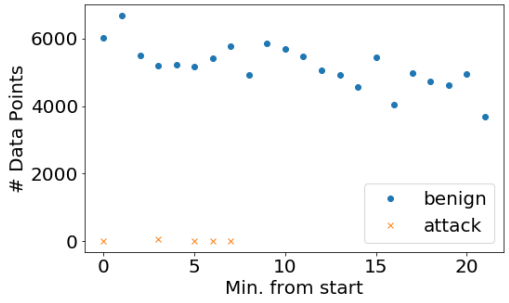
NO_AGGR	AGGR_3M
# Benign: 554281	# Benign: 108412
# Attack: 4630 (0.83%)	# Attack: 42 (0.04%)

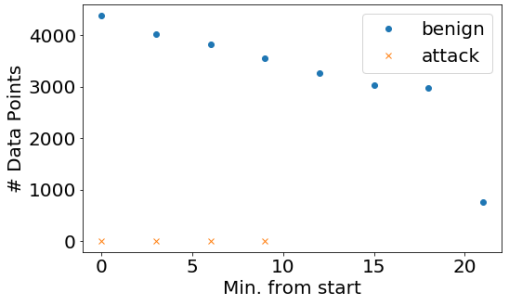




7 Sogou

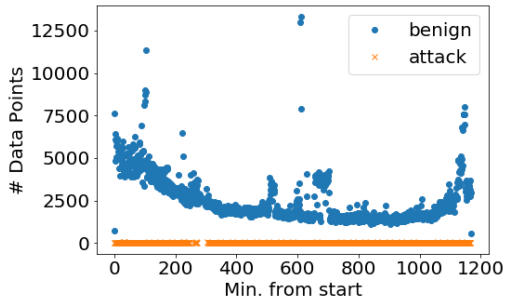
NO_AGGR	AGGR_3M
# Benign: 114011	# Benign: 25800
# Attack: 63 (0.06%)	#Attack: 4 (0.02%)

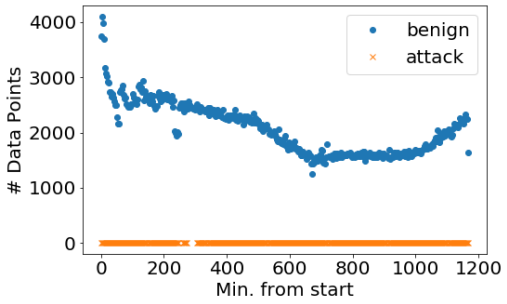




8 Murlo

NO_AGGR	AGGR_3M
# Benign: 2948040	# Benign: 805205
# Attack: 6127 (0.21%)	# Attack: 376 (0.05%)



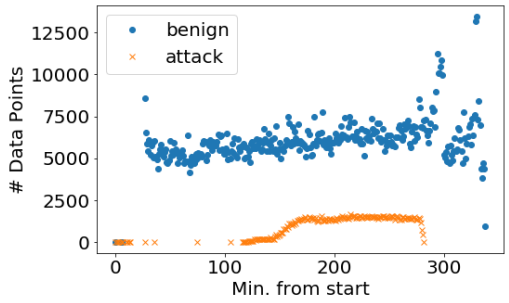


CTU-13 attack scenarios
Continued on next page

9 Neris

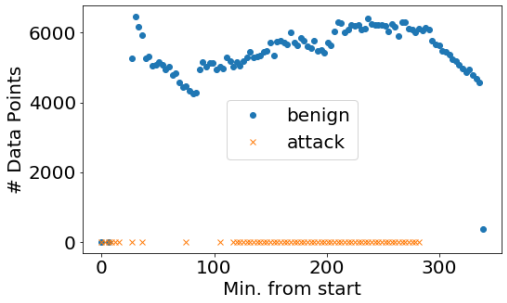
NO_AGGR

Benign: 1902500
Attack: 184987 (8.86%)



AGGR_3M

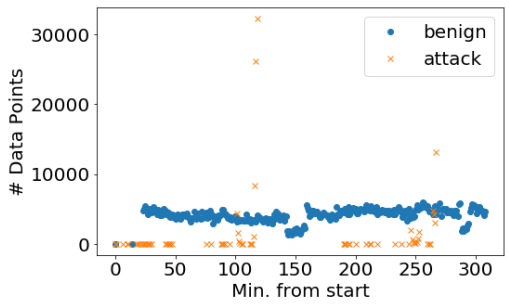
Benign: 574339
Attack: 504 (0.09%)



10 Rbot

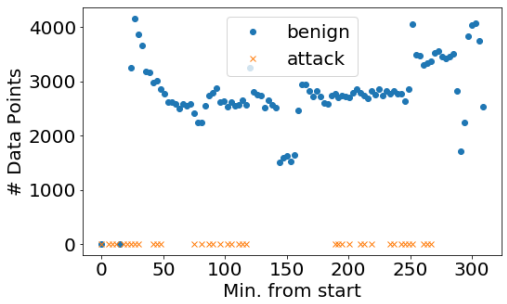
NO_AGGR

Benign: 1203423
Attack: 106352 (8.12%)



AGGR_3M

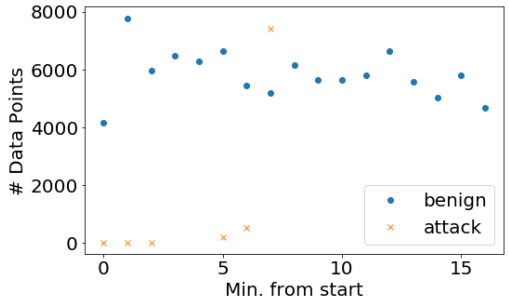
Benign: 271622
Attack: 234 (0.09%)



11 Rbot

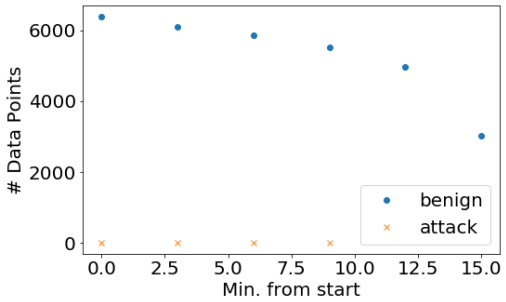
NO_AGGR

Benign: 99081
Attack: 8164 (7.61%)



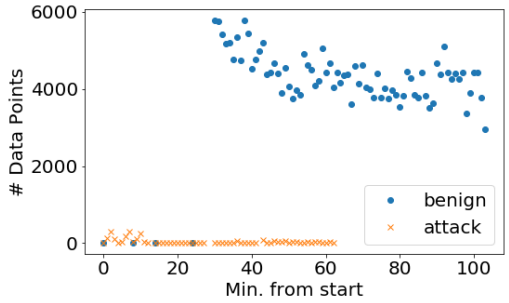
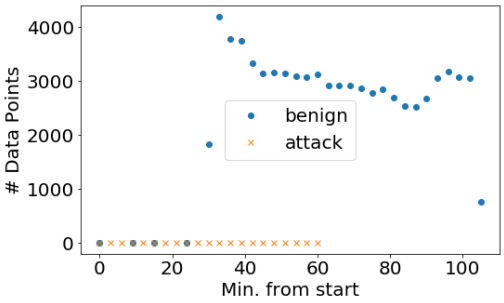
AGGR_3M

Benign: 31853
Attack: 8 (0.03%)



CTU-13 attack scenarios
Continued on next page

12 NSIS.ay

NO_AGGR	AGGR_3M
# Benign: 323298	# Benign: 76484
# Attack: 2168 (0.67%)	# Attack: 66 (0.09%)
	

13 Virut

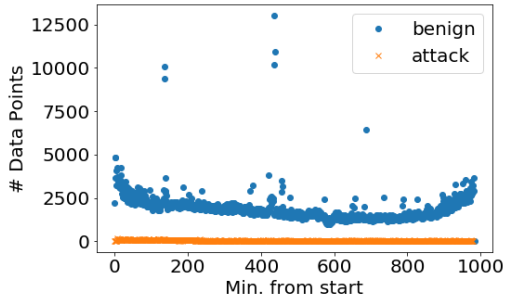
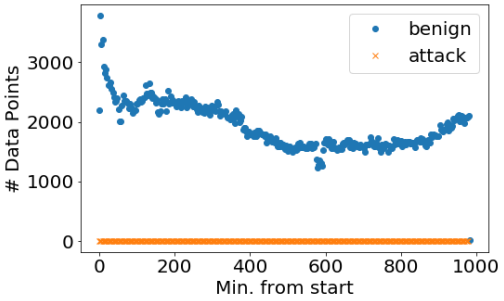
NO_AGGR	AGGR_3M
# Benign: 1885094	# Benign: 633803
# Attack: 40003 (2.08%)	# Attack: 327 (0.05%)
	

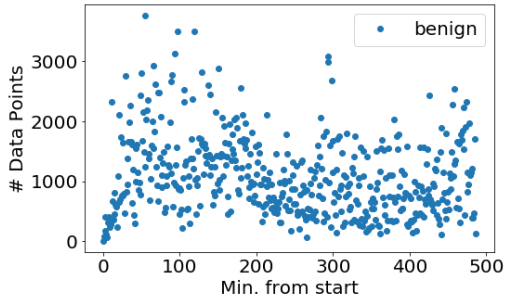
Table B1: Overview of CTU-13 attack scenarios

CICIDS 2017

1 Benign

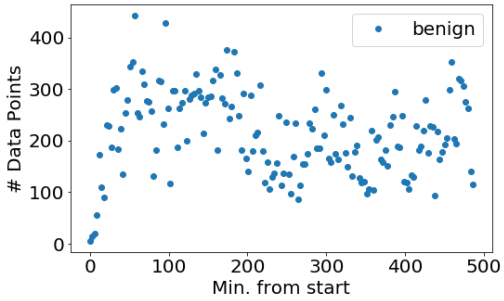
NO_AGGR

Benign: 529918
Attack: 0 (0.00%)



AGGR_3M

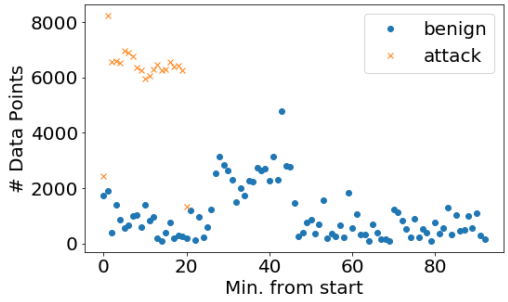
Benign: 35017
Attack: 0 (0.00%)



2 DDoS

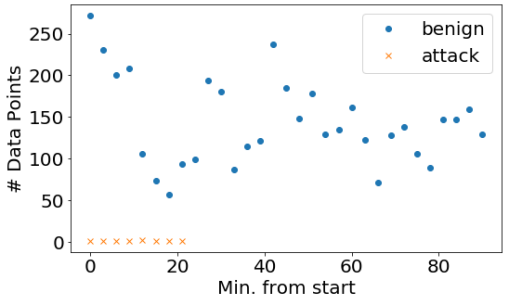
NO_AGGR

Benign: 97718
Attack: 128027 (56.71%)



AGGR_3M

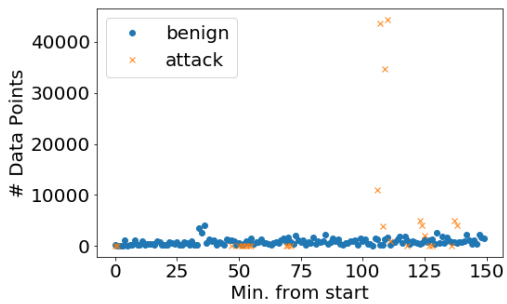
Benign: 4451
Attack: 9 (0.20%)



3 Port Scan

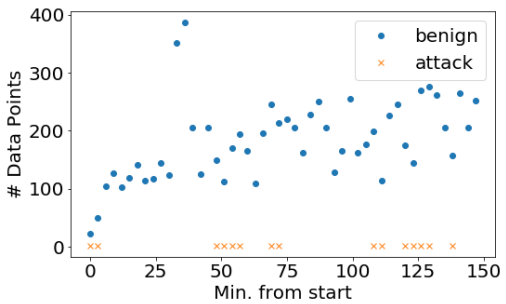
NO_AGGR

Benign: 127537
Attack: 158930 (55.48%)



AGGR_3M

Benign: 9158
Attack: 15 (0.16%)

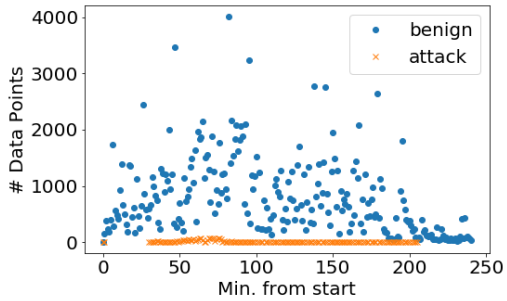


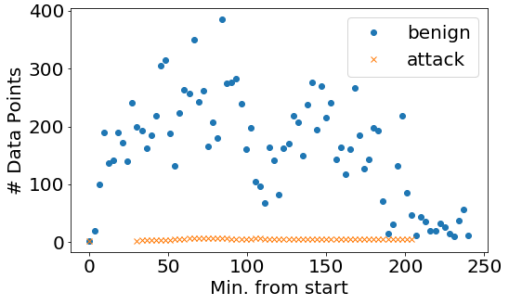
CICIDS 2017 attack scenarios

Continued on next page

4 Botnet

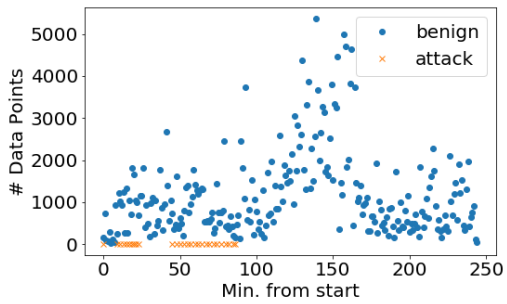
NO_AGGR	AGGR_3M
# Benign: 189067	# Benign: 12802
# Attack: 1966 (1.03%)	# Attack: 291 (2.22%)

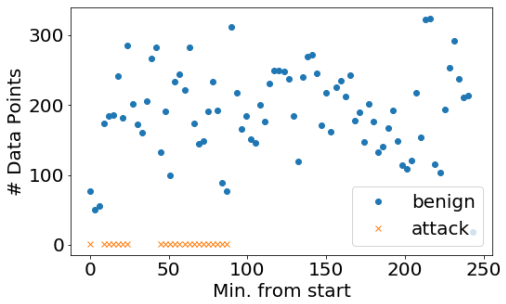




5 Infiltration

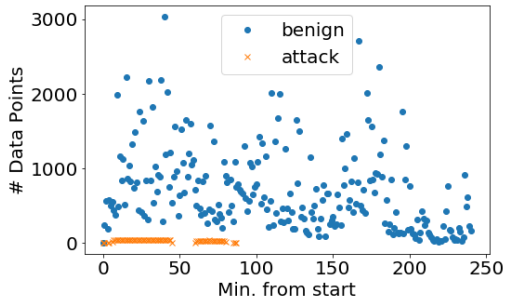
NO_AGGR	AGGR_3M
# Benign: 288565	# Benign: 15622
# Attack: 36 (0.01%)	# Attack: 22 (0.14%)

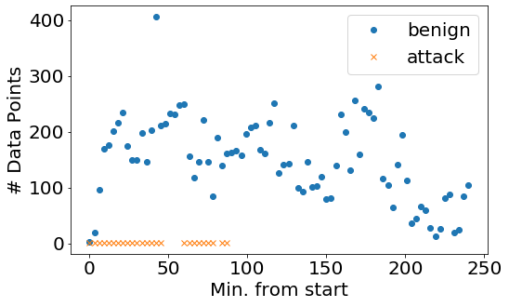




6 Web

NO_AGGR	AGGR_3M
# Benign: 168186	# Benign: 12159
# Attack: 2180 (1.28%)	# Attack: 25 (0.21%)





CICIDS 2017 attack scenarios

Continued on next page

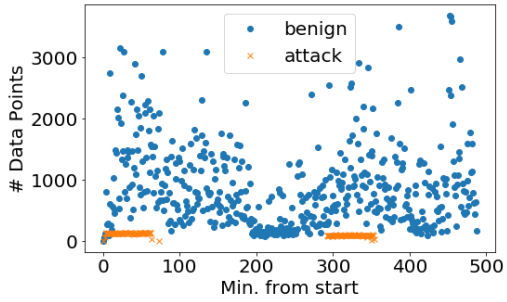
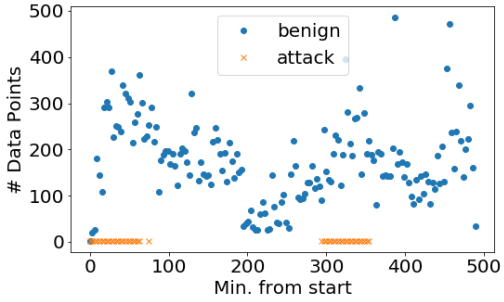
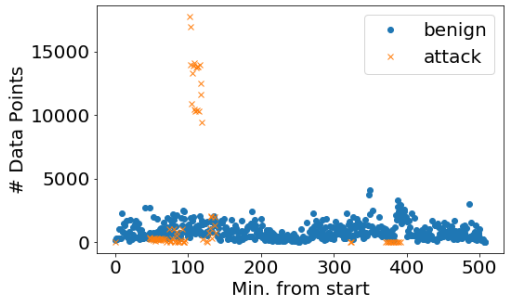
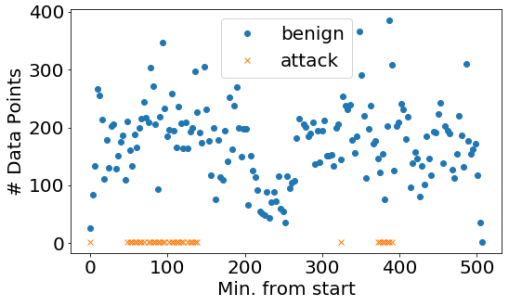
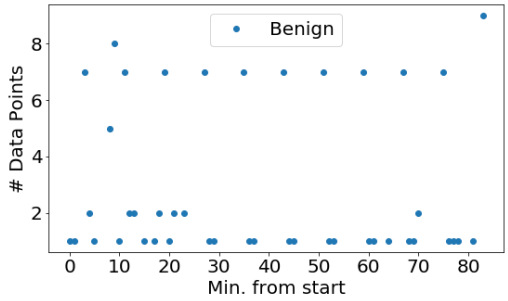
7 Brute-force	
NO_AGGR	**AGGR_3M**
# Benign: 432074 # Attack: 13835 (3.10%)	# Benign: 28660 # Attack: 44 (0.15%)
	
8 DoS	
NO_AGGR	**AGGR_3M**
# Benign: 440031 # Attack: 252672 (36.48%)	# Benign: 29413 # Attack: 37 (0.13%)
	

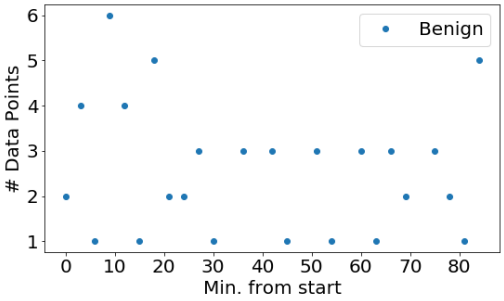
Table B2: Overview of CICIDS 2017 attack scenarios

IoT-23

1 Benign (7)

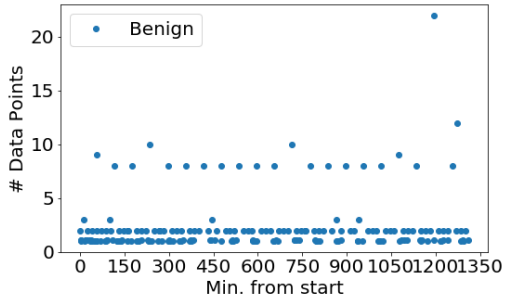
NO_AGGR		AGGR_3M	
# benign:	130	# benign:	62
# attack:	0 (0.00%)	# attack:	0 (0.00%)

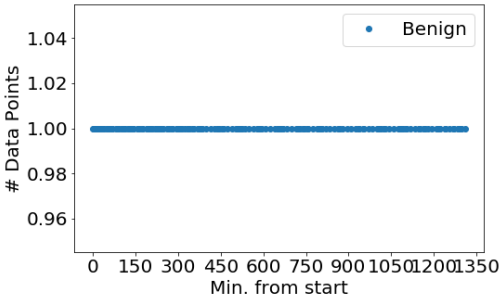




2 Benign (4)

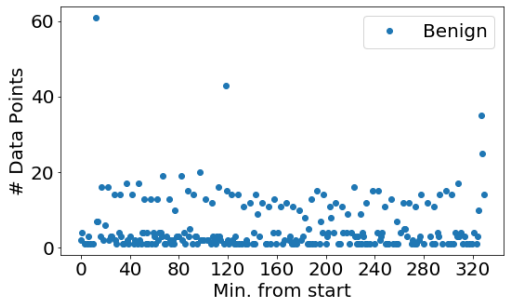
NO_AGGR		AGGR_3M	
# benign:	452	# benign:	171
# attack:	0 (0.00%)	# attack:	0 (0.00%)

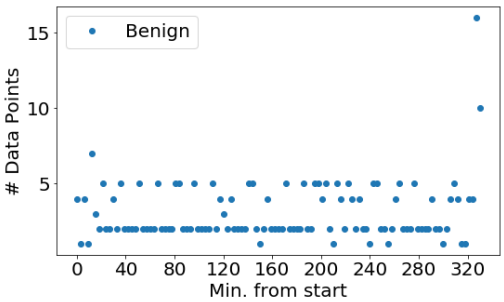




3 Benign (5)

NO_AGGR		AGGR_3M	
# benign:	1374	# benign:	340
# attack:	0 (0.00%)	# attack:	0 (0.00%)



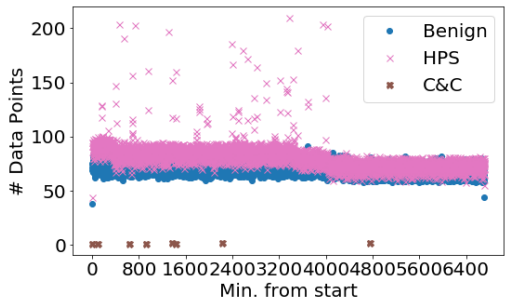


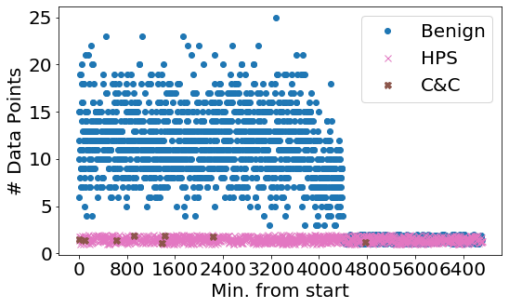
IoT-23 attack scenarios

Continued on next page

4 Mirai (1)

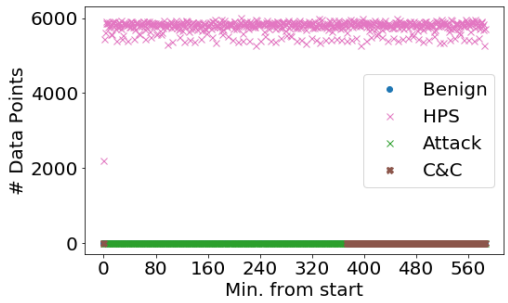
NO_AGGR		AGGR_3M	
# benign:	469275	# benign:	17419
# attack:	539473 (53.48%)	# attack:	2239 (11.39%)
- # HPS:	539465 (53.48%)	- # HPS:	2231 (11.35%)
- # C&C:	8 (0.00%)	- # C&C:	8 (0.04%)

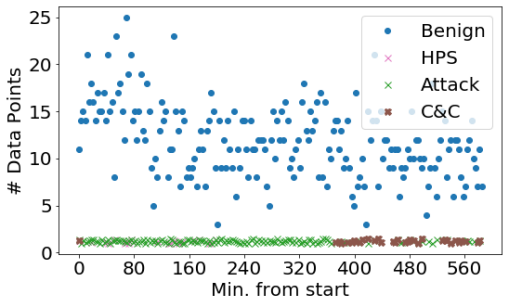




5 Mirai (48)

NO_AGGR		AGGR_3M	
# benign:	3734	# benign:	2335
# attack:	3390604 (99.89%)	# attack:	196 (7.74%)
- # HPS:	3387007 (99.78%)	- # HPS:	20 (0.79%)
- # Attack:	2752 (0.08%)	- # Attack:	144 (5.69%)
- # C&C:	845 (0.02%)	- # C&C:	32 (1.26%)

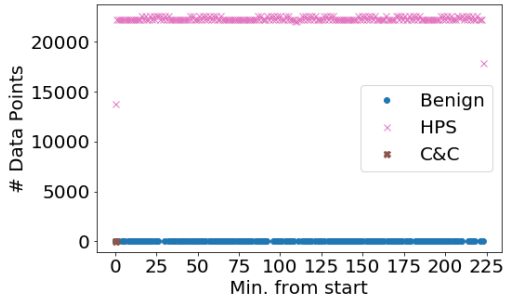


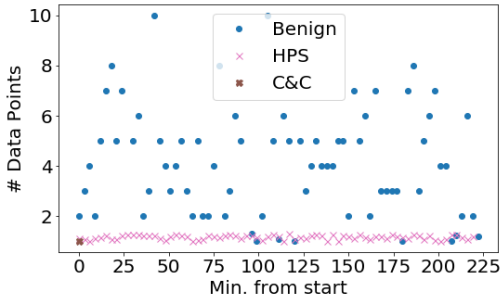


Continued on next page

6 Mirai (52)

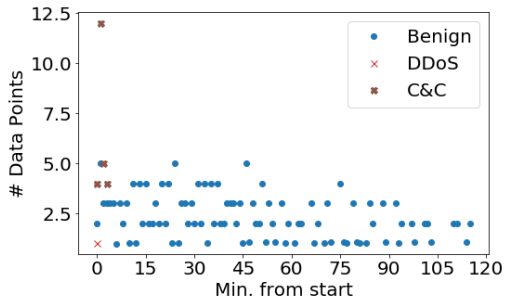
NO_AGGR		AGGR_3M	
# benign:	405	# benign:	302
# attack:	4999595 (99.99%)	# attack:	75 (19.89%)
- # HPS:	4999576 (99.99%)	- # HPS:	74 (19.63%)
- # C&C:	19 (0.00%)	- # C&C:	1 (0.27%)

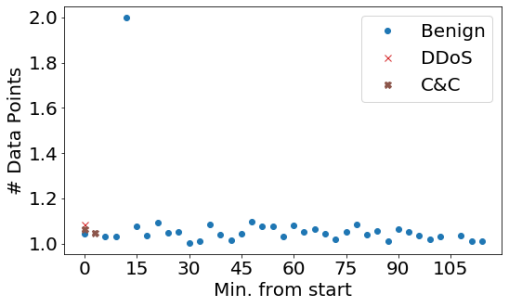




7 Mirai (44)

NO_AGGR		AGGR_3M	
# benign:	211	# benign:	38
# attack:	26 (10.97%)	# attack:	3 (7.32%)
- # DDoS:	1 (0.42%)	- # DDoS:	1 (2.44%)
- # C&C:	25 (10.55%)	- # C&C:	2 (4.88%)

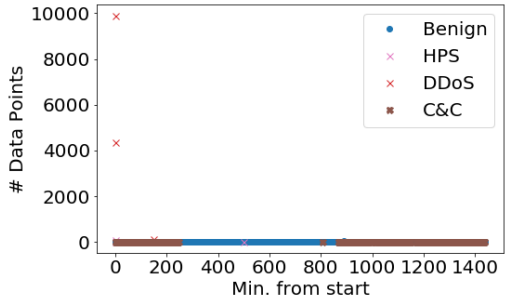


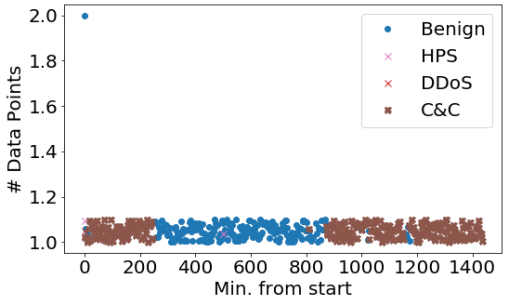


Continued on next page

8 Mirai (34)

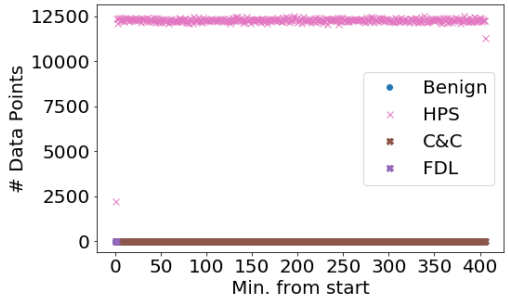
NO_AGGR		AGGR_3M	
# benign:	1923	# benign:	212
# attack:	21222 (91.69%)	# attack:	261 (55.18%)
- # HPS:	122 (0.53%)	- # HPS:	2 (0.42%)
- # DDoS:	14394 (62.19%)	- # DDoS:	1 (0.21%)
- # C&C:	6706 (28.97%)	- # C&C:	258 (54.55%)

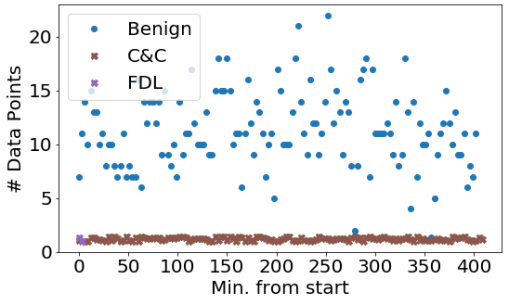




9 Mirai (49)

NO_AGGR		AGGR_3M	
# benign:	3433	# benign:	1528
# attack:	4996567 (99.93%)	# attack:	138 (8.28%)
- # HPS:	4994775 (99.90%)	- # HPS:	1 (0.06%)
- # C&C:	1778 (0.04%)	- # C&C:	136 (8.16%)
- # FDL:	14 (0.00%)	- # FDL:	2 (0.12%)





Continued on next page

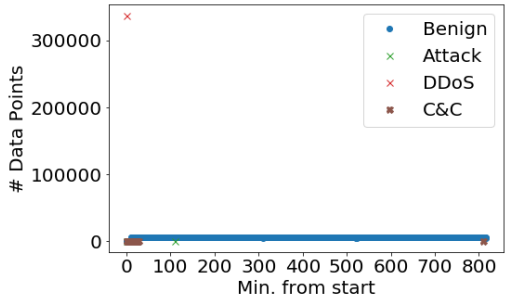
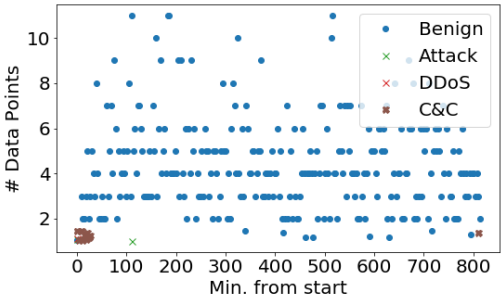
10 Mirai (35)	
NO_AGGR	**AGGR_3M**
# benign: 4662273	# benign: 1190
# attack: 337727 (6.75%)	# attack: 14 (1.16%)
- # Attack: 3 (0.00%)	- # Attack: 2 (0.17%)
- # DDoS: 337632 (6.75%)	- # DDoS: 1 (0.08%)
- # C&C: 92 (0.00%)	- # C&C: 11 (0.91%)
	

Table B3: Overview of IoT-23 attack scenarios

C Selected Hyperparameters

This section presents the selected hyperparameter settings for the models. We perform hyperparameter selection separately for split-at-random and split-by-scenario benchmarks.

RANDOM SPLIT				
Model	**Parameter**	**CTU-13**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	15-5	20-10-10-5	20-10-5
	encoding_dim	3	3	3
	batch_size	1024	1024	64
VAE	*encoder_net*	15-5	20-10-10-5	20-10-5
	latent_dim	2	2	5
	batch_size	1024	1024	64
AEGMM	*encoder_net*	15-5	15-5	-
	encoding_dim	1	1	-
	n_gmm	2	4	-
	batch_size	2048	2048	-
IF	*num_estimators*	200	50	50
MAH	*num_components*	4	2	2
SCENARIO SPLIT				
MODEL	**Parameter**	**CTU-13**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	20-10-5	20-10-10-5	15-5
	encoding_dim	3	3	3
	batch_size	64	64	1024
VAE	*encoder_net*	20-10-10-5	20-10-5	15-5
	latent_dim	1	1	2
	batch_size	1024	1024	1024
AEGMM	*encoder_net*	20-10-10-5	20-10-5	15-5
	encoding_dim	1	1	1
	n_gmm	2	4	4
	batch_size	2048	2048	2048
IF	*num_estimators*	200	100	50
MAH	*num_components*	4	2	2

Table C1: Selected hyperparameters CTU-13

RANDOM SPLIT				
Model	**Parameter**	**CICIDS 2017**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	60-30-10	60-30-10	60-30-10
	encoding_dim	3	3	3
	batch_size	1024	64	64
VAE	*encoder_net*	60-30-10	60-30-10	60-30-10
	latent_dim	5	2	1
	batch_size	64	64	1024
AEGMM	*encoder_net*	50-10	50-10	60-30-10
	encoding_dim	1	1	1
	n_gmm	2	4	4
	batch_size	2048	2048	2048
IF	*num_estimators*	100	200	100
MAH	*num_components*	2	8	4
SCENARIO SPLIT				
Model	**Parameter**	**CICIDS 2017**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	50-10	60-30-10	60-30-10
	encoding_dim	3	3	3
	batch_size	1024	64	64
VAE	*encoder_net*	50-10	60-30-10	60-30-10
	latent_dim	5	5	2
	batch_size	64	64	64
AEGMM	*encoder_net*	60-30-10	60-30-10	60-30-10
	encoding_dim	1	1	1
	n_gmm	2	2	4
	batch_size	2048	2048	2048
IF	*num_estimators*	100	200	50
MAH	*num_components*	2	4	4

Table C2: Selected hyperparameters CICIDS 2017

RANDOM SPLIT				
Model	**Parameter**	**IoT-23**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	50-10	50-10	50-10
	encoding_dim	3	3	3
	batch_size	1024	1024	64
VAE	*encoder_net*	60-30-10	60-30-10	50-10
	latent_dim	1	1	1
	batch_size	64	64	64
AEGMM	*encoder_net*	-	50-10	50-10
	encoding_dim	-	1	1
	n_gmm	-	2	2
	batch_size	-	2048	2048
IF	*num_estimators*	100	100	200
MAH	*num_components*	4	4	8
SCENARIO SPLIT				
Model	**Parameter**	**IoT-23**		
		NO_AGGR	**AGGR_M**	**AGGR_3M**
AE	*encoder_net*	60-30-10	50-10	50-10
	encoding_dim	3	3	3
	batch_size	1024	64	64
VAE	*encoder_net*	50-10	60-30-10	50-10
	latent_dim	1	1	1
	batch_size	1024	64	64
AEGMM	*encoder_net*	-	60-30-10	50-10
	encoding_dim	-	1	1
	n_gmm	-	4	4
	batch_size	-	2048	2048
IF	*num_estimators*	100	50	100
MAH	*num_components*	8	4	4

Table C3: Selected hyperparameters IoT-23

D Performance per Scenario on Split-at-Random Benchmarks

This section presents the performance-per-scenario results obtained on the split-at-random benchmark. First, we select the models that achieve the highest overall F_1-score (one for every possible combination of { AE, VAE, AEGMM, IF, MAH } and { NO_AGGR, AGGR_M, AGGR_3M }). For each selected model, we evaluate its performance at detecting various attack scenarios.

Id	Name	Aggr.	Model				
			AE	**VAE**	**AEGMM**	**IF**	**MAH**
1	Neris	NO	0.1837	**0.2146**	0.1464	0.1342	0.1307
		1M	0.842	**0.8929**	0.5042	0.7217	0.6719
		3M	0.902	0.9073	-	0.8288	**0.933**
2	Neris	NO	**0.6336**	0.5995	0.0945	0.4922	0.3373
		1M	0.8841	**0.9217**	0.4678	0.8007	0.7534
		3M	0.9252	0.9198	-	0.8718	**0.9386**
3	Rbot	NO	0.2924	**0.3435**	0.0044	0.2345	0.2827
		1M	0.5821	**0.7329**	0.5738	0.4375	0.2417
		3M	0.6478	0.6644	-	0.4983	**0.6904**
4	Rbot	NO	0.659	**0.686**	0.2338	0.4992	0.6488
		1M	0.5674	0.569	**0.6097**	0.6092	0.4458
		3M	0.5123	0.4776	-	**0.6555**	0.3807
5	Virut	NO	0.6326	0.4171	0.5397	**0.7628**	0.6352
		1M	0.9242	0.8545	0.3847	**0.9744**	0.9562
		3M	0.9149	0.9144	-	**0.9891**	0.9828
6	Menti	NO	0.8877	**0.8974**	0.0371	0.756	0.8156
		1M	0.9315	**0.9513**	0.8124	0.882	0.9103
		3M	**0.9671**	0.9655	-	0.9326	0.95
7	Sogou	NO	0.6005	0.6032	**0.732**	0.6788	0.0725
		1M	0.3266	0.0	0.32	**0.7272**	0.6642
		3M	0.0	0.0	-	0.499	**0.5037**
8	Murlo	NO	0.5119	**0.5494**	0.0563	0.359	0.362
		1M	0.1885	**0.2063**	0.1389	0.1707	0.1786
		3M	0.2859	0.3007	-	0.2353	**0.323**
9	Neris	NO	**0.3291**	0.3214	0.1473	0.2399	0.1993
		1M	0.849	**0.8607**	0.3569	0.7472	0.7889
		3M	0.8714	0.8831	-	0.8189	**0.9035**
10	Rbot	NO	0.7305	**0.7575**	0.3677	0.5678	0.6524
		1M	**0.6943**	0.6732	0.6757	0.6583	0.2476
		3M	0.675	0.6486	-	**0.7013**	0.2034
11	Rbot	NO	0.9667	**0.9706**	0.8793	0.9463	0.9672
		1M	0.608	0.6123	0.6024	**0.9194**	0.603
		3M	0.5997	0.5995	-	**0.7185**	0.444
12	NSIS.ay	NO	0.0633	0.0615	**0.6418**	0.2645	0.4784
		1M	0.552	0.5156	0.5452	**0.607**	0.2997
		3M	**0.5504**	0.5501	-	0.3187	0.2428
13	Virut	NO	**0.5733**	0.3614	0.0654	0.4435	0.4225
		1M	**0.796**	0.4063	0.3185	0.5942	0.4462
		3M	0.8554	**0.8842**	-	0.7377	0.7061

Table D1: Test F_1-score per scenario CTU-13

Id	Name	Aggr.	Model				
			AE	**VAE**	**AEGMM**	**IF**	**MAH**
2	DDoS	NO	0.5345	0.5798	0.5978	0.2248	**0.5981**
		1M	**0.988**	0.9617	0.9872	0.9173	0.9688
		3M	0.9589	**0.978**	0.9227	0.934	0.9264
3	Port Scan	NO	0.0005	0.0045	**0.1469**	0.0003	0.0014
		1M	0.6533	0.6511	0.5815	0.5204	**0.6581**
		3M	0.4319	**0.4498**	0.3702	0.3989	0.3728
4	Botnet	NO	0.0286	0.0349	**0.0391**	0.0236	0.0254
		1M	0.0747	0.1957	0.1007	**0.4218**	0.0092
		3M	0.3763	0.1636	0.0532	**0.7073**	0.0
5	Infiltration	NO	**0.846**	0.8446	0.7835	0.6519	0.7057
		1M	**0.8874**	0.8834	0.8579	0.7592	0.8759
		3M	0.9351	**0.9404**	0.8757	0.8335	0.8391
6	Web	NO	0.0178	**0.0687**	0.0175	0.0176	0.0164
		1M	0.0	**0.7554**	0.0	0.7113	0.0
		3M	**0.8108**	0.6307	0.0	0.8049	0.0
7	Brute-force	NO	0.0012	0.0012	**0.0053**	0.0	0.0001
		1M	0.0	**0.8008**	0.322	0.5523	0.0
		3M	0.8497	**0.8857**	0.3452	0.6955	0.01
8	DoS	NO	0.589	**0.5999**	0.5234	0.5929	0.5397
		1M	**0.758**	0.7217	0.5952	0.5445	0.4427
		3M	0.8032	**0.8669**	0.5195	0.6657	0.5381

Table D2: Test F_1-score per scenario CICIDS 2017

Name	Aggr.	Model				
		AE	**VAE**	**AEGMM**	**IF**	**MAH**
C&C	NO	0.2702	0.2722	-	**0.291**	0.2815
	1M	**0.9219**	0.8732	0.8165	0.9188	0.7475
	3M	0.8211	0.6694	0.9108	**0.9314**	0.7174
Horizontal Port Scan	NO	0.1573	0.0206	-	**0.5589**	0.0067
	1M	0.9762	0.9762	0.9762	0.4004	**1.0**
	3M	**0.9772**	0.8867	0.8824	0.6657	0.6645
DDoS	NO	0.9754	**0.9856**	-	0.8144	0.6569
	1M	0.6053	**0.7759**	0.4884	0.5924	0.6455
	3M	0.64	0.381	0.0	0.25	**0.9412**
Attack	NO	0.0382	**0.0418**	-	0.0382	0.0152
	1M	0.6206	0.5114	0.5568	0.625	**0.6368**
	3M	0.4925	0.6819	0.675	**0.6842**	0.4718
File Download	NO	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
	1M	NA	NA	-	NA	NA
	3M	**1.0**	**1.0**	**1.0**	0.6667	**1.0**

Table D3: Test F_1-score per scenario IoT-23

E Performance per Scenario on Split-by-Scenario Benchmarks

This section presents the performance-per-scenario results obtained on the split-by-scenario benchmark. The idea is the same as described in the previous section.

Id	Name	Aggr.	Model				
			AE	**VAE**	**AEGMM**	**IF**	**MAH**
1	Neris	NO	**0.0772**	0.0067	0.0012	0.0093	0.001
		1M	0.0226	**0.0367**	0.0085	0.0179	0.0073
		3M	0.0123	**0.0198**	0.0046	0.0068	0.0002
2	Neris	NO	**0.2315**	0.0312	0.0012	0.0457	0.045
		1M	0.0242	**0.0383**	0.0083	0.0195	0.0059
		3M	0.0132	**0.022**	0.0052	0.0074	0.0002
6	Menti	NO	**0.1194**	0.0018	-	0.0008	0.0015
		1M	0.0298	**0.052**	0.0313	0.0261	0.0103
		3M	0.0204	**0.0357**	0.0099	0.0112	0.0008
8	Murlo	NO	0.0003	0.0004	0.0001	**0.0011**	0.0
		1M	0.0112	0.0137	0.01	0.0087	**0.0141**
		3M	0.0077	**0.0112**	0.005	0.005	0.001
9	Neris	NO	**0.2229**	0.039	0.0003	0.0891	0.0829
		1M	0.1108	**0.1804**	0.0166	0.0977	0.0306
		3M	0.0696	**0.1083**	0.033	0.0411	0.0018

Table E1: Test F_1-score per scenario CTU-13

Id	Name	Aggr.	Model				
			AE	**VAE**	**AEGMM**	**IF**	**MAH**
4	Botnet	NO	0.017	0.0185	**0.0929**	0.0188	0.0028
		1M	0.0937	0.1626	0.0301	**0.1997**	0.1655
		3M	0.185	0.1955	0.047	**0.3123**	0.2722
5	Infiltration	NO	**0.0129**	0.0099	0.0006	0.0005	0.0031
		1M	**0.25**	0.0654	0.2431	0.0367	0.0361
		3M	0.168	0.1597	**0.2483**	0.078	0.0238
7	Brute-force	NO	0.0004	0.0095	0.099	**0.1072**	0.0
		1M	0.0765	**0.1132**	0.0	0.0443	0.0
		3M	0.1305	**0.1338**	0.0669	0.0404	0.0224
8	DoS	NO	0.7549	**0.7633**	0.7216	0.5826	0.3749
		1M	**0.2651**	0.0802	0.1684	0.0463	0.0268
		3M	0.1231	0.1138	**0.1402**	0.0685	0.0211

Table E2: Test F_1-score per scenario CICIDS 2017

Name	Aggr.	Model				
		AE	**VAE**	**AEGMM**	**IF**	**MAH**
C&C	NO	**0.4786**	0.4565	-	0.4662	0.24
	1M	0.5121	0.5121	0.561	**0.684**	0.5132
	3M	0.5139	0.5139	0.5144	**0.6701**	0.2663
Horizontal Port Scan	NO	**0.7269**	0.6804	-	0.7039	0.0037
	1M	**0.75**	0.7	0.6667	0.2165	0.5
	3M	**0.6667**	0.5	0.0	0.0204	0.0
DDoS	NO	0.5449	0.5824	-	0.5319	**0.9649**
	1M	**0.5013**	0.3346	0.1679	0.0033	0.0013
	3M	**0.5038**	0.3372	0.0038	0.009	0.0038
Attack	NO	0.2979	**0.3501**	-	0.2979	0.2989
	1M	**0.5025**	0.5025	0.5019	0.4961	0.5025
	3M	0.5059	0.5059	**0.5076**	0.4892	0.5076
File Download	NO	0.0081	**0.0085**	-	0.0081	0.0081
	1M	0.0	0.0	**1.0**	0.4	0.0
	3M	**1.0**	**1.0**	0.6667	0.3636	0.0

Table E3: Test F_1-score per scenario IoT-23