UNIVERSITÄT BONN

Topology-based Intrusion Detection for Homogeneous IoT Devices

Reproducable Generation of Diverse IDS Datasets

Master's Thesis

by

Christian Bungartz
2898807

in fulfillment of requirements for degree
Master of Science (M.Sc.)

submitted to
Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik IV
Arbeitsgruppe für IT-Sicherheit

in degree course
Computerscience (M.Sc.)

First Supervisor: Dr. Felix Boes
University of Bonn

Second Supervisor: Prof. Dr. Matthew Smith
University of Bonn

Bonn, May 4, 2022

Abstract

Topological data analysis (TDA) allows the characterization of a given space's topology. This enables anomaly detection based on structural differences between various point clouds, representing, for example, the behavior of a host. As the given characterization is a very robust representation of the underlying data, it holds many benefits for classification tasks in various areas of IT security. Particularly interesting are its use cases within the context of ChangeDrift. ChangeDrift is an abstract concept regarding the detection of anomalous behavior of any sort within a system of homogeneous nodes. An area where this becomes increasingly more important is IoT devices, especially in non-individual areas of use. Here homogeneity of large sets of nodes is typically given and can thus be utilized to facilitate intrusion detection.

This work focuses on intrusion detection for a set of homogeneous devices. Accordingly, the goal is to utilize deviations in the devices' behavior to enable detection of various attacks. This includes both deviations with respect to device state, but also device behavior over an extended period of time. To this end, various approaches to include topological data analysis in the detection process are presented and subsequently evaluated. TDA is introduced at three different stages, namely to achieve inter-device sensor data fusion of more elaborate sensors, feature extraction from device behavior over time and data classification. The results are very promising. Especially the feature extraction is heavily outperforming a more traditional approach.

Contents

1 Introduction

The increasing complexity and diversity of cyber-attacks increase the need for intrusion detection systems (IDS) to guarantee information integrity. We classify IDS's based on their detection approach into two categories. Namely, these are knowledge-based and anomaly-based systems. Knowledge-based intrusion detection relies on known attack signatures to discover them in network or system data. On the other hand, the anomaly-based approach models normal, benign user system behavior and detects deviations as abnormal and thus potentially malicious. Thus, an anomaly-based approach is more flexible and capable of detecting even previously unknown attacks, with the tradeoff being a worse false positive rate.

A special area of intrusion detection is concerned with homogeneous devices. Here the goal is to detect deviations from expected behavior or device state, utilizing the other devices in the network. Consequently, anomaly-based detection is the appropriate choice. In this area, the vast majority of algorithms belong to data analysis and machine learning, which can be separated into the categories unsupervised and supervised. For the purpose of anomaly detection, a model has to be generated that — based on the deviations from expected behavior — is capable of establishing the attack type. To minimize the false positive rate in anomaly-based IDS, the quality of this model is paramount.

1.1 Motivation

Intrusion detection for IoT devices is gaining increasingly more relevance as attacks against IoT devices continue to rise. In the first six months of 2021 alone, Kaspersky detected 1.5 billion attacks based on telemetry from their IoT honeypots, which equals an increase of over 100% compared to the number of attacks in the latter half of 2020 [Sea21]. As IoT devices rise in prominence, this illustrates the importance of countermeasures against these types of attacks. Particularly interesting are larger, non-individual areas of application. These typically feature multiple near-identical devices performing a very similar task, which enables us to

utilize the behavior of multiple devices simultaneously to detect misbehavior of a subset of these.

This problem is a specialized version of the ChangeDrift concept, which concerns anomaly detection based on divergent behavior. In particular, we are interested in comparing — possible complex — device states (as reported by a collection of sensors) either directly in the form of a state analysis or over an extended time period as behavior analysis. A concept that promises to perform the required information extraction is topological data analysis, specifically persistent homology. Topological data analysis (TDA) denotes the analysis of the topological properties of a given space based on a finite sample of data points. In contrast to geometric properties (e.g., quantify a space based on distance, shape, size), topological properties are invariant to homeomorphism (i.e., they allow stretching and bending of a given shape) [Car21].

A key concept in TDA is persistent homology (PH). PH characterizes the structural properties of a given data set [ZJZ21]. This characterization can be expressed as the lifetime of i-dimensional holes subject to the filtration distance (cf. chapter 4) [Car21]. These topological features and their lifetimes can then be utilized to generate numeric feature vectors in the case of behavior analysis or to calculate distances between more complex sensor values. This enables a robust description of the difference between multiples space that — at least theoretically — appears to provide several advantages for classification. The invariance to homeomorphism inherited by the PH-based approaches allows us to devise very robust classifiers against small variations of input data (e.g., sensor noise). Accordingly, PH-based feature extraction is a very promising research topic not just for our specific use case, but in multiple fields of IT security.

Detection of deviations in homogeneous IoT devices, however, comes with a number of challenges. Most prominent is the question of how exactly to encapsulate device behavior (state) deviations (feature extraction) with high information retention to allow for good class separability. Once a method is established, models have to be trained and evaluated with respect to not only their classification capabilities but also compatibility with the respective encapsulation method. On top of that, in the field of IoT devices, we encounter a very high variance of attack and usage scenarios, which — together with the feature extraction — gives rise to the need for specialized datasets. This can only be met insufficiently by adjusting already existing datasets. Thus, our secondary goal is the conception and implementation of a tool facilitating the reproducible creation of IDS datasets given a user-defined scenario.

This demand for such a tool is not unique to our aforementioned use case. In any field concerning classification or anomaly detection, the quality of data is paramount to the quality

of the final product. Two factors primarily decide the quality of a dataset. On the one hand, is the quality of the individual samples. In the field of intrusion detection, this includes criteria like the selection of the attacks, the realism of the benign data, and the chosen scenario, to name a few. Secondly, the reproducibility of the dataset has to be considered. This describes the documentation of the dataset generation process as well as the extensibility and adaptability of the finished dataset. Former is defined by the ease with which a third party is able to understand both the actual process of generating as well as the reasoning behind the decisions made regarding criteria like the choice of attacks. Latter is once again concerned with the documentation — this time primarily with the documentation of the scenario — but also potentially with toolsets made available (e.g., the CICFlowMeter in the case of the CICIDS dataset) [SHG18]. Good reproducibility allows the generation of an improved or specialized version of the given dataset.

1.2 Research Goal

The goal of the master's project is two-fold. **Primary goal** is the evaluation of topological data analysis (TDA) facilitated intrusion detection for homogeneous IoT devices at various stages of the detection process. Namely, these stages are sensor data fusion, feature extraction, and classification. We place a special emphasis on integrating the homogeneous nature of the devices in the detection process. This concept of utilizing divergent behavior in a set of homogeneous nodes is referred to as ChangeDrift. We argue that TDA not only is able to aid in this task, but provide advantages over a more traditional approach. To this end, we also evaluate our TDA-based approaches in direct comparison to a previous work exploring intrusion detection for homogeneous devices [Ten21].

Secondary goal is the conception and implementation of a tool allowing the reproducible generation of IDS datasets given a user-defined scenario, without sacrificing flexibility and diversity of the generated datasets. We assess the usability and functionality of this tool by recreating two popular IDS datasets, CICIDS and ADFA-LD [SHG18; CH13]. Additionally, we utilize the newly generated toolset to create a couple of datasets focusing on the field of anomaly-based intrusion detection for homogeneous IoT devices. These are then utilized in the evaluation of the first research question.

1.3 Thesis Structure

We first introduce the tool by detailing its different stages and modular systems and motivate the design choices, as it is crucial for the evaluation of our primary research question (cf. chapter 2). The evaluation of the tool, by recreating the CICIDS2017 and ADFA-LD dataset, follows in chapter 3. Afterward, we introduce topological data analysis based feature extraction and classification methods (cf. chapter 4). We then detail the experiment setup concerning the evaluation of TDA-based intrusion detection for homogeneous devices (cf. chapter 5). Here we discuss the different scenarios and resulting datasets as well as the classification procedure and motivate all our decisions. The evaluation is given in chapter 6 together with an in-depth description of the TDA-methods applications. The thesis finishes with the conclusion (cf. chapter 7) and possible future work (cf. section 7.3).

2 Reproducible Generation of IDS Datasets

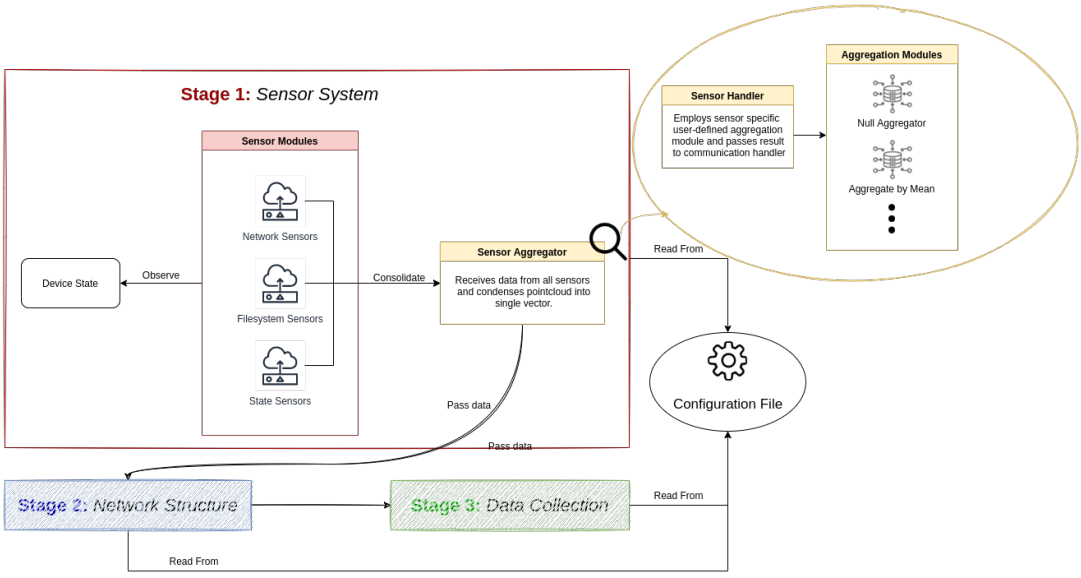


Figure 1: *Tool, first stage.*

To achieve the reproducible generation of IDS datasets, we introduce a new tool. The tool is separated into three stages, each containing a set of modular systems. This allows high flexibility as well as easy extensibility without sacrificing usability or expressive power. The three stages are the sensor system, the network structure, and the data collection. The selection of the different modules and the wiring between them is performed using a user-defined configuration file. This enables reproducibility and flexibility with respect to the data collection (outside influences like attacks are not handled by the tool). In the following, we will discuss each stage individually while detailing the respective modular systems and reasoning behind our choices. The first stage including the sensor and aggregator modules are presented in section 2.1, the second stage and the network modules are given in section 2.2, with section 2.3 discussing the third stage and the service modules. Afterward, we motivate the sensor selection in section 2.4 and present a minimum working example in section 2.5.

Table 1: *Available state sensors.*

Sensor ID	Task	Captured Measurements
State Sensors		
cpu_load	Measures CPU load.	CPU utilization as float.
memory_utilization	Measures memory utilization	Memory load as float.
process_info	Collects user-defined information about a process defined by its process identifier.	User-defined subset of process metrics: ▪ Number of reads ▪ Number of writes ▪ Number of bytes read ▪ Number of bytes written ▪ Number of bytes passed to read/pread system call ▪ Number of bytes passed to write/pwrite system call ▪ Number of voluntary context switsches ▪ Number of involuntary context switches ▪ Time spent in user mode ▪ Time spent in kernel mode ▪ Time all children spent in user mode ▪ Time all children spent in kernel mode ▪ Number of spawned threads ▪ Nice value ▪ Number of file descriptors
new_process_info	Collects user-defined information about all processes spawned during the measurement period.	Dictionary of *process_info* identified by PID.
syscall	Collects system calls that occurred during the measurement interval.	List of system calls.

Table 2: *Available network sensors.*

Sensor ID	Task	Captured Measurements
Network Sensors		
network_activity	Collects statistics about all active connections.	User-defined subset of connection metrics: ▪ Destination IP ▪ Destination Port ▪ Source Port ▪ PID ▪ Type (TCP/UDP/Other) ▪ Status
network_traffic	Captures all network traffic on user-selected interface.	PCAP of observed traffic.

2.1 Tool: First Stage

The first stage is the sensor system. Figure 1 shows a schema of its layout. The purpose of this stage is to handle the measurement collection. To this end, two modular systems are employed. Namely, these are the sensor modules and the aggregator modules. Sensor modules observe the device state and produce a stream of sensor data of varying formats and lengths, while aggregators condense these data streams into a user-defined format. We differentiate between three types of sensors.

2.1.1 Available Sensor Modules

Firstly, we have the **network sensors**. This category includes all sensors that in some form monitor outgoing and incoming network traffic (cf. Table 2). In its current state, the pipeline features two sensors in this category; a network packet sniffer, capturing all traffic on the selected interface, and a network activity sensor, which keeps track of active connections (e.g., by destination IP/port, process identifier, and other.).

The second category is the **file system sensors**, which monitor file system changes. At the time of this master thesis, the tool features a total of two file system sensors. These are a disk utilization sensor as well as a file access sensor. The prior collects data about the disk utilization employing statistics like busy-time, read/written bytes, read/write time, and other of the likes (cf. Table 3). The latter observes which files, in particular, are interacted with. To

Table 3: *Available file system sensors.*

Sensor ID	Task	Captured Measurements
Filesystem Sensors		
accessed_files	Measures which files are accessed during the measurement interval (default directories: ['/home/', '/etc/', '/bin/', '/root/', '/sbin/', '/usr/']).	List containing names of the files.
disk_utilization	Measures disk utilization.	User-defined subset of disk IO metrics: ▪ Busy time ▪ Number of bytes read ▪ Number of bytes written ▪ Number of reads ▪ Number of writes ▪ Time spent reading ▪ Time spent writing ▪ Number of merged reads ▪ Number of merged writes

this end, it keeps track of all files accessed in specified directories during the measurement period. We choose the default directories based on our specific use case and in particular, its attacks. Hence, they most likely require adaption to the usage scenario of the user.

Lastly, there is the category of **state sensors**. As state sensor qualifies any sensor that directly observes the system state. We utilize sensors measuring CPU load and memory utilization. Furthermore, we provide a sensor collecting information about user-defined processes employing statistics like read/written bytes, voluntary/involuntary context switches, CPU time, and the number of spawned threads, among others (cf. Table 1). This sensor can be extended using the *new_process_info* sensor, which collects information about all processes spawned during the measurement interval. Lastly, the tool also provides a system call sensor, which can be utilized to collect system call traces.

2.1.2 Available Aggregator Modules

We differentiate between two important time intervals during the measurement collection. Namely, these are the **sensor interval** and the **aggregation interval**. The sensor interval is the sampling rate of the individual sensors. The aggregator module system serves to process the individual sensor data and combine the streams from the different sensors into a single unified data stream. Here the aggregation interval defines the number of sensor measurements combined in each aggregation step. To this end, each sensor is assigned a specific user-defined aggregator module that aggregates all measurements observed during the aggregation interval into a single one. These — per sensor aggregated — sensor values are then combined into a single list. The results from the aggregation system are passed on to the communication handler in the second stage of the tool.

The simplest aggregator that is part of the tool is the null-aggregator. This module passes on any data its receives without alteration. More involved is the function-aggregator. This aggregator applies a user-defined function to the data stream and returns the results. Only standard statistical quantities are implemented in the tool's current state (mean, median, minimum, and maximum). Furthermore, we have a map-aggregator. This module maps the sensor data to an output value employing a user-defined map. The aggregator is the so-called concat-aggregator, which handles sensors returning lists or sets (e.g., the accessed files sensor) and combines all sensor values into a single list.

Table 4: *Available aggregators.*

Aggregator ID	Task
concat	Combines all sensor values into a single list.
map	Maps sensor data to new value employing a user-defined map.
null	Passes sensor data on without any modification.
Function Aggregators	
max	Passes on the maximum sensor value.
mean	Calculates the mean of the sensor values.
median	Calculates the median of the sensor values.
min	Passes on the minimum sensor value.

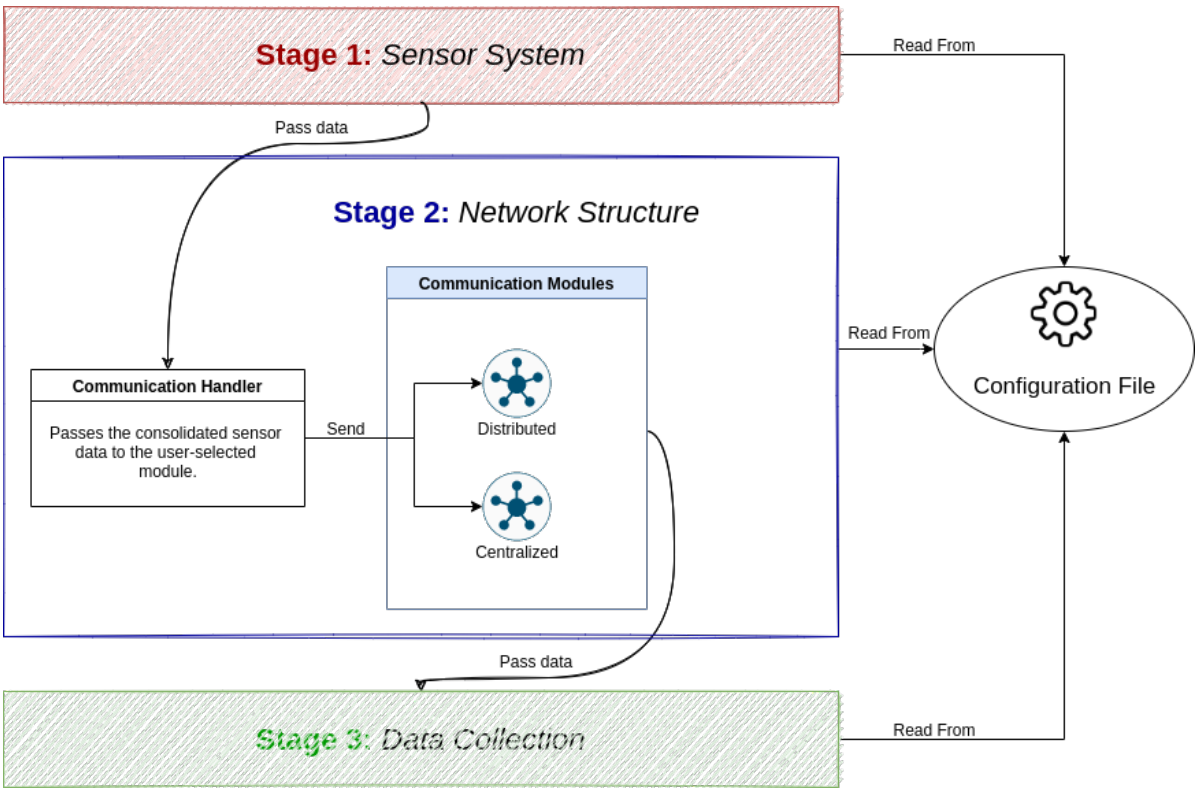


Figure 2: *Tool, second stage.*

2.2 Tool: Second Stage

The second stage handles the aggregated sensor data distribution throughout the network. At the time of writing, the tool supports two different network architectures. These are a centralized architecture with a central managing server and a peer-to-peer one, as found in, for example, a mobile ad-hoc network. In the centralized one, each client sends its data to the respective central server, with all data collection and management tasks being delegated to it. The peer-to-peer module features a broadcast data distribution with all clients receiving data from all other clients. This is necessary to mimic a distributed real-world scenario. A potential IDS in a distributed network operates by delegating the detection process to the respective participating clients. Thus, a significant amount of data needs to be passed around, which will influence the sensors themselves, and therefore we need to emulate this behavior to get an accurate dataset.

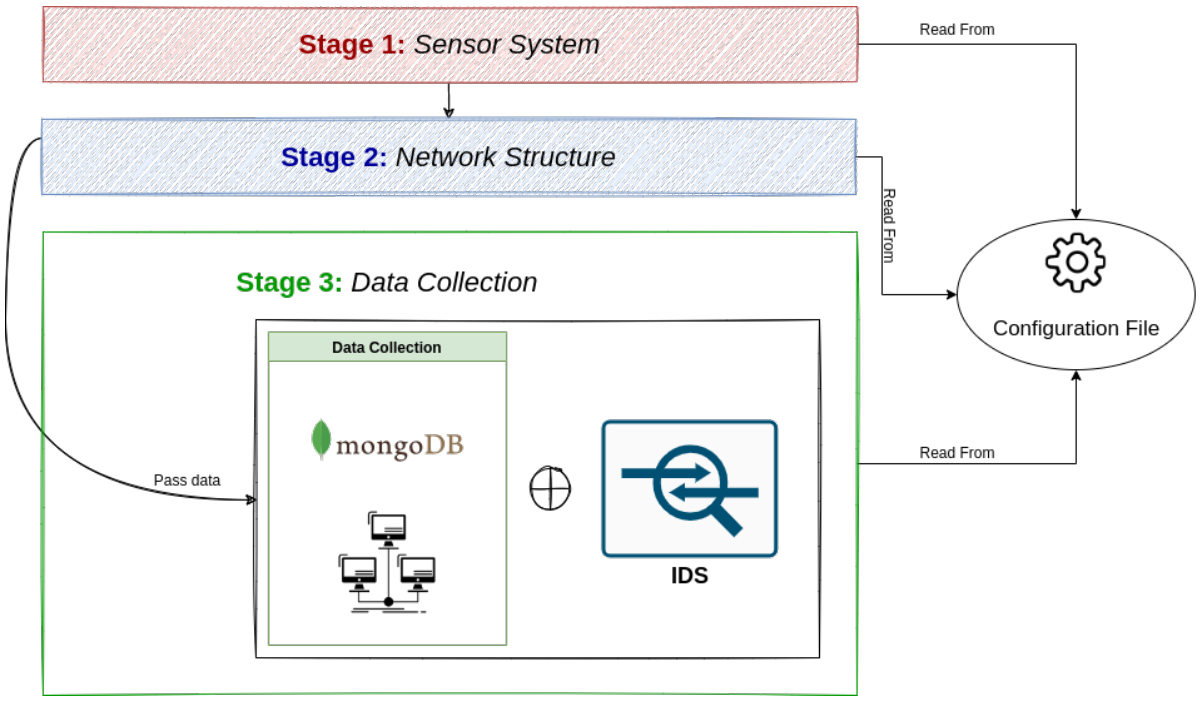


Figure 3: *Tool, third stage.*

2.3 Tool: Third Stage

Data collection and a potential IDS sit at the third stage of the tool. The data collection process can be augmented using a system of service modules. For the purpose of this thesis, we developed three different modules. Since data collection is very dependent on the network structure, the service modules are exclusive to the different architectures. We utilize a central MongoDB server to collect the sensor data from all clients for the centralized case. The respective service module thus, handles the translation of the aggregated data of all sensors to a respective MongoDB insertion.

For the distributed architecture, we offer two service modules. Both utilize local storage on a subset of the devices denoted as "scribes" to persist the sensor data. Specifically, these are modules for storing network captures as a PCAP file and a service module storing sensor data as a local JSON file. All clients marked as scribes will store all sensor data from all clients. We offer the possibility of assigning multiple scribes for ease of extensibility given a distributed network setup where the data is not broadcasted but rather multicasted.

2.4 Sensor Selection

The selection of sensors implemented for this first version of the tool is motivated by its intended usage throughout this master thesis. Accordingly, the selection is made based on the two datasets to be recreated as well as the IoT datasets with respect to our scenario and hardware choices. Especially important in the context of the IoT datasets is the attack selection, as it directly dictates the anomalous behavior we aim to capture. A detailed overview of the scenarios and attack selection, as well as our reasoning behind their choice, is given in section 5.3. In the following, we first discuss the sensor choices motivated by the two reproduced datasets (cf. section 2.4.1) and afterward focus on the IoT scenarios (cf. section 2.4.2).

2.4.1 Reproduction Datasets motivated sensor choices

The reproduction of the datasets leaves not much choice for decisions, as they specifically require a certain set of sensors. The two datasets we replicate are the **ADFA-LD** and the **CICIDS2017** datasets [CH13; SHG18]. **ADFA-LD** is a host-based intrusion detection dataset that contains system calls observed on the host both during normal operation and while under attack [CH13]. Consequently, its reproduction demands a sensor capable of capturing system calls (sensor ID: *syscall*). To this end, we utilize the auditd framework. Auditd is Linux's audit system residing in the kernel and collecting in rule-files specified kernel events into log files [Ma+18]. It is the most widely used auditing system for Linux and thus a good choice for the task of collecting the required system call traces [Ma+18]. Our client is configured to access the log files, preprocess them (i.e., extracting the traces and system calls the user is interested in), and pass this information on.

Contrary to ADFA-LD **CICIDS2017** is a network-based intrusion detection dataset containing captured network traffic characterized by 80 features [SHG18]. To extract the feature set, they employ CICFlowMeter, a software by the Canadian Institute for Cybersecurity, taking PCAPs as input [Hab+17]. The software is freely available and, as such, is utilized by us to extract the same feature set. Consequently, the requirement for the sensor is the capture of network traffic as PCAP (sensor ID: *network_traffic*). To this end, we use Scapy. Scapy is a Python module that allows the capture and manipulation of packets on a specified interface [Bio22].

2.4.2 IoT specific sensor choices

Regarding the IoT datasets, the sensor choices are primarily based on the attack selection as they dictate what behavior difference we attempt to capture. As mentioned in our research goal 1.2 this part of our research extends and adapts on a previous work introducing an IDS specifically for intrusion detection for homogeneous devices (in this work referred to as T-ChangeDrift) [Ten21]. Due to this, part of the sensors is adopted from this reference work. These are *cpu_load*, *memory_utilization* and *accessed_files*. To capture the CPU and RAM utilization, we make use of Psutil, a Python library allowing the capture of process and system information [Rod22]. The file access is collected employing Watchdog, a file system event monitoring library [Man22].

While this selection is sufficient for T-ChangeDrift, we also aim to detect attacks that operate much more subtle. This gives rise to the need for sensors capable of capturing system behavior in greater detail. To this end, we extend on one sensor already present in the reference work as well as introduce an additional three of our own.

With regard to network connections, T-ChangeDrift only collects the IP addresses on connected hosts. Justified is their decision by focusing on (D)DoS attacks, which are clearly identifiable by the connection to typically not accessed hosts. However, this is not sufficient for us. Given our intention of dataset generation instead of IDS implementation, we use an abstraction of the hosts' addresses to prevent our classification algorithms from simply learning the respective attacker addresses. Furthermore, we think focussing on (D)DoS leads to the dismissal of a number of attacks that have high relevance but are not necessarily identifiable by the host address alone. Namely, these fall under the categories of reconnaissance and infiltration, which very well might be performed from a system that is known to the IoT devices but was somehow compromised and is now used for lateral movement within the network [Osm+20]. Relying purely on host addresses can lead to this attack going unnoticed. Consequently, we also capture source and destination port, PID, type, and status for a more fine-grained differentiation between expected and unexpected connections (sensor ID: *network_activity*). To achieve this, we use Psutil [Rod22].

We concentrate on disk utilization and process information as far as new sensors are concerned. The goal of the former is to capture potential file manipulations of files also legitimately accessed by the system (sensor ID: *disk_utilization*). This is a scenario that is not sufficiently met by just the *accessed_files* sensor on its own. Latter is justified by the introduction of attacks that are less taxing on the device's hardware. An example are attacks utilizing

the Metasploit framework or also, to some extent, botnets (cf. section 5.3). Here we want to be able to capture the system state fine-grained enough to be able to correctly associate malicious actions as abnormal behavior even if no external connection is established. To this end, we employ a process information sensor (sensor ID: *process_info*). Since we do not know the respective process identifiers in advance, we extend this sensor to capture information about all processes spawned during the measurement period (sensor ID: *new_processs_info*). We once again employ Psutil to collect the necessary information [Rod22].

2.5 Tool: Minimum Working Example

```yaml
metadata:
  clients: 2
  client_type: heterogeneous
  time:
    - 26.11.21 12:00:00 - 26.11.21 13:15:00
  client_meta:
    pi_0:
      os: Ubuntu 20.04
      ip: 192.168.1.193
      platform: manylinux2014_x86_64
      python_version: 3.9.0
    pi_1:
      os: Ubuntu 18.04
      ip: 192.168.1.214
      platform: manylinux2014_x86_64
      python_version: 3.9.0

# Stage 1: Sensors and Aggregators
sensor_system:
  clients:
    - client_id: pi_0
      sync_interval: 10

      sensors:
      - type: cpu_load
        parameter:
          interval: 0.2
        aggregator: mean
        aggregator_parameter: {}
```

```
30
31          - type: memory_usage
32            parameter:
33              interval: 0.2
34            aggregator: mean
35            aggregator_parameter: {}
36        - client_id: pi_1
37          sync_interval: 5
38
39          sensors:
40          - type: cpu_load
41            parameter:
42              interval: 0.1
43            aggregator: max
44            aggregator_parameter: {}
45
46          - type: memory_usage
47            parameter:
48              interval: 0.1
49            aggregator: median
50            aggregator_parameter: {}
51
52    # Stage 2: Network
53    network:
54      clients:
55        - client_id: pi_0
56          type: p2p
57          service: local_json  # Stage 3: Data Collection
58          network_parameter:
59            scribes: [pi_0]
60          service_parameter:
61            save_location: results/
62        - client_id: pi_1
63          type: p2p
64          service: local_json  # Stage 3: Data Collection
65          network_parameter:
66            scribes: [pi_0]
67          service_parameter:
```

```
68          save_location: results/
```

Listing 2.1: *Sample YAML, accepted by our tool.*

The tool consists of two parts, the generator and the client created by the generator. The generator expects a configuration file in YAML format (by default called *config.yaml*). This configuration contains all necessary information for the generator as well as additional information that is primarily useful to the respective end-users recreating the experiment setup. A configuration that can be employed to generate a simple dataset is given in Listing 2.1. This configuration is designed for the usage with two Raspberry Pis running Ubuntu 20.04 and 18.04, respectively, and captures their CPU and volatile memory utilization over a period of 75 minutes. Both devices utilize different aggregation methods and measurement intervals (both sensor and aggregator). The data is distributed using a peer-to-peer network and stored by the Pi with ID "*pi_o*" in a JSON file on disk. The configuration consists of three parts, which we discuss in the following sections.

2.5.1 Configuration

Metadata contains general information about the experiment setup as well as client generation. The possible parameters are *clients*, *client_type*, *time* and *client_meta*. *Clients* define the number of clients while *client_type* defines their type. We differentiate between homogeneous and heterogeneous clients. Listing 2.1 shows the setup for a number of heterogeneous clients. In this case, we need to specify each client configuration separately. This includes the *client_meta*, which defines meta-information of the clients. In this example we are working with heterogeneous clients — identified by their respective client ID —, thus there is a respective setup for each individual client ID. Homogeneous clients do not require a per-client setup, and thus, the respective parameters can be set globally for all clients. In this case, the client meta denotes the clients' operating system, the respective platform, the provided Python version, as well as their IP. The measurement period is also defined as part of the meta-data.

The main part of the client configuration is done employing the **sensor system**. The sensors system denotes the first stage of the client (cf. Figure 1) and thus includes the sensors themselves as well as the respective aggregator. Each component is identified by its string identifier (cf. tables 3, 2, 1). Since we are utilizing a heterogeneous set of clients, each client has its own sensor configuration. Using the keys *"parameter"* and *"aggregator_parameter"*, user parameters can be passed to the respective modules. Especially important are the parameter *"interval"* (configured for each sensor individually) and *"sync_interval"* (configured globally

for the entire sensor system). *"sync_interval"* sets the synchronization interval for all sensors in seconds. At the end of every interval, the collected sensor data is passed to the respective aggregators, accumulated, and then passed on as one data stream to the network manager. The sensor parameter *"interval"* sets the sampling rate of the sensor itself.

Lastly, there is the **network** configuration. Here the user defines the data collection as well as the inter-client communication. To this end, we differentiate between *type* and *service*. *Type* defines the network architecture and is modified by the *"network_parameter"*, corresponding to stage two of the tool (cf. Figure 2). Currently, we support a peer-to-peer ("p2p") and centralized ("centralized") network architecture. *Service* on the other hand determines the data collection and is modified by the *"service_parameter"* and thus is representing stage three (cf. Figure 3). Currently supported are two types of local storage (*"local_json"* and *"local_pcap"*; former as distributed storage) and MongoDB as the second centralized option.

2.5.2 Deployment

Once the configuration file is completed, the generator can be run, which will result in a number of files being generated in the *data* directory of the tool. These are a README file and the client software for each client, given as a tarball compressed with Bzip2. The README details the experiment setup. This includes details about the client software and how it can be utilized, but more importantly, it specifies how both clients and networks should be set up. These include client IPs, operating systems, tasks, and software requirements that can not be automatically set up and configured.

Each client package identifies itself by its client ID and is customized for that specific client. The adjustments done by the generator to each client package primarily concern the configuration and setup of the respective client. However, certain modules are also activated respectively deactivated based on the experiment setup. This deactivation allows for a less bloated client with consequently fewer dependencies. The installation of the client involves unpacking the client followed by running the setup script (*setup.py*). After this, the client can be started using *python collector_cli.py* and will automatically start collecting sensor data at the beginning of the measurement period and gracefully exit at the end of it.

3 Tool Evaluation

To evaluate the tool's capabilities, we recreate two common IDS datasets. Namely, these are the CICIDS2017, and the ADFA-LD datasets [CH13; SHG18]. The CICIDS2017 is a modern network-based IDS dataset, while the ADFA-LD dataset contains system-call traces. This allows us to showcase the tool in two distinct scenarios. Particularly, this is a setup where we are required to monitor multiple clients at once and store potentially large amounts of sensor data (CICIDS2017) versus a straightforward setup with only one device but more complex sensor data (ADFA-LD). In the following, we evaluate the two recreated datasets by employing classifications methods from relevant reference papers and comparing the results to the ones they achieved. To this end, for each recreation we first introduce the dataset (cf. section 3.1 and 3.2), then give an overview of the experiment setup (cf. section 3.1.1 and 3.2.1) and afterward discuss the classification results as well as notable observations (cf. section 3.1.2 and 3.2.2).

3.1 On the Recreation of ADFA-LD

Software	Version
Apache	2.4.41
PHP	7.4.3
MySQL	8.0.28
TikiWiki	23.1
OpenSSH	8.2
Vsftpd	3.0.3

Label	Count
Training	833
Validation	4373
Hydra-SSH	148
Hydra-FTP	162
Meterpreter	125
Meterpreter Java	75
Add User	91

Figure 4: *Software versions (left) and distribution of system call traces (right) [CH13].*

The ADFA-LD dataset was created in 2013 by G. Creech, and J. Hu to provide a dataset for host-based intrusion detection [CH13]. It is a set of system call traces collected over several hours on a Linux machine running Ubuntu 11.04. The original dataset features a total of six attacks. Namely, these are [CH13]:

1. Hydra-FTP
2. Hydra-SSH
3. Adduser
4. Java-Meterpreter
5. Meterpreter
6. Webshell

Due to an insufficient description of the "Webshell" attack, we only recreate the first five attacks. However, given our objective of evaluating the usability and adaptability of our tool in the context of creating a HIDS dataset and not the perfect reproduction of ADFA, this does not pose a problem.

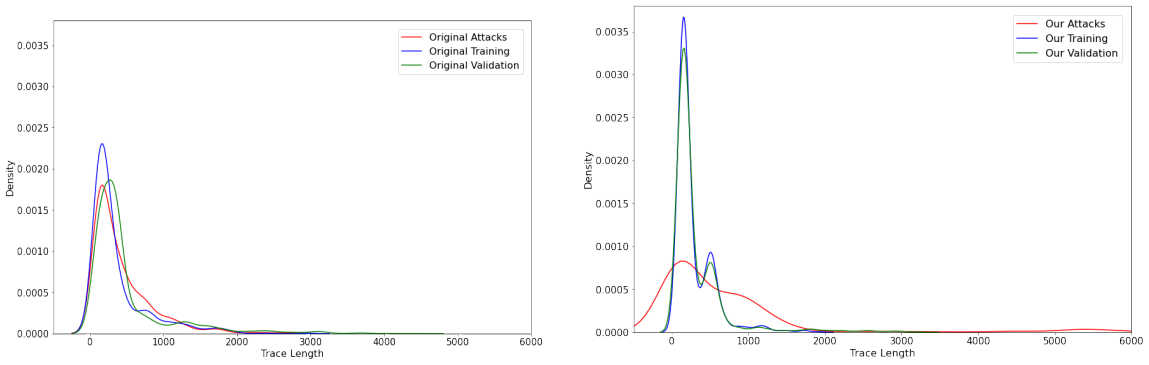
3.1.1 Experiment Setup

The experiment is conducted employing an aarch64 machine (Raspberry Pi 4 B) running Ubuntu Server 20.04 kernel version 5.4.00-1050-raspi. The underlying hardware is a four-core Cortex-A72 processor with 8GB of memory. Notable is our decision to deviate both in terms of system architecture and software version. Specifically, in terms of software and operating system, we utilize the latest stable versions instead of the legacy versions while still using the same software as the original dataset. We argue that this decision is justified due to primarily our own research goal, which is the evaluation of the data collection scenario, not the perfect reproduction of the dataset. Furthermore, all attacks evaluated do not directly exploit a software vulnerability. Instead, they either exploit software configurations (e.g., no limitations on the login attempts for SSH/FTP) or are self-contained malware. Thus, we do not expect meaningful differences based on software versions other than the benign background noise. However, since the description of the benign system trace generation is severely lacking, we expect major differences in this field either way.

The software providing benign background data as well as being attacked by the attacker is given in Figure 4[CH13]. The captured sensor data is written to disk, and the corresponding file is retrieved after the experiment conclusion. After collection, we preprocess the data as described in the original paper. This includes the removal of system call traces with a corresponding file size outside the interval 300B to 6KB for training and 300B to 10KB for validation instances. Additionally, we sample a subset from the traces to match the exact number of traces in the original dataset, as no attack time is provided (cf. Figure 4).

We generate benign data by performing multiple typical user actions, which can be attributed to three categories. The first category is creating, deleting, and editing articles on the hosted TikiWiki. In addition, we send/receive, move and delete files in the home directory using both FTP and SSH. Lastly, we execute standard shell commands over SSH to view/modify the content of files in the home directory, show active network connections, show directory contents and write new files.

3.1.2 Classification Results



(a) Original traces

(b) Reproduced traces

Figure 5: *Kernel density estimation of the trace lengths for both our samples and the original dataset (y-axis of graph 5(b) is cut off at 6000 for uniformity between the two subfigures).*

Figure 5 shows the kernel density estimation of the system-call trace lengths. The plots show significant differences between the two datasets. Most notable are the differences in similarity of validation and training data as well as the difference in the shape of the probability distribution of both attack traces. Both of these can be attributed to decisions made by us to substitute missing information about the original experiment setup. The probability distributions of both validation and attack data match almost perfectly in our data, while there is a clear shift in the original. This shift is not caused by the extended upper limit of the file size interval as it is centered around a trace length of 500 system calls. Instead, it indicates a clear behavior shift in benign user behavior that, unfortunately, is never mentioned in the original paper. Thus, it is impossible for us to recreate this, and instead, we perform the same (similar) benign actions during the entire measurement interval. The second major difference can be observed in the attack trace length. The reference paper does not specify an upper size bound for the attack traces. As such, our data features traces up to a length of over 12000 system calls (cut off in the plot for readability and uniformity between subfigures). This is not

observable in the original data. Otherwise, the trace length distribution matches the original one, with the majority being in the length interval $[0, 2000]$.

For the evaluation, we utilize the results obtained by Miao Xie and Jiankun Hu in their paper "Evaluating Host-Based Anomaly Detection Systems: A Preliminary Analysis of ADFA-LD" which was published shortly after the dataset itself [XH13]. They employ a very rudimentary classification approach based on the anomalous nature of the attack instances. To this end, a variation of the k-nearest-neighbors algorithm is employed. We choose this paper due to the simplicity of the classification approach (enabling a better comparison) as well as the close proximity of its publication to the release of the dataset. In the following, we introduce the classification approach utilized by Miao Xie and Jiankun Hu. Consequently, all following refers to their respective paper [XH13].

Miao Xie and Jiankun Hu employ frequency-based approaches to extract feature vectors from the system call lists. In its simplest form, a feature vector is generated by counting for each possible system call ($t \in \mathbf{T}$) how many times it can be observed in the trace. Each entry is then normalized by the total number of system calls in the trace resulting in a vector of weights. This weight calculation can be expressed using equation 4.1 where s is the trace, t are the respective system calls and $\bar{w}_s(t)$ the resulting weights [XH13].

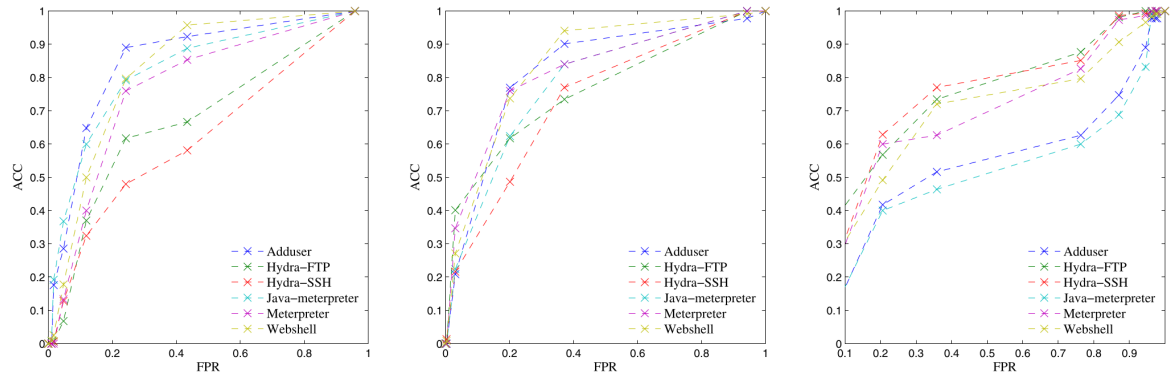
$$\bar{w}_s(t) = \frac{f_s(t)}{|s|} \tag{3.1}$$

Other approaches employed in the reference work are inverse document frequency and its standardized variant (cf. equations 4.2 and 4.3)[XH13]:

$$\hat{w}_s(t) = \bar{w}_s(t) * \log\left(\frac{1}{\bar{w}_s(t)}\right) \tag{3.2}$$

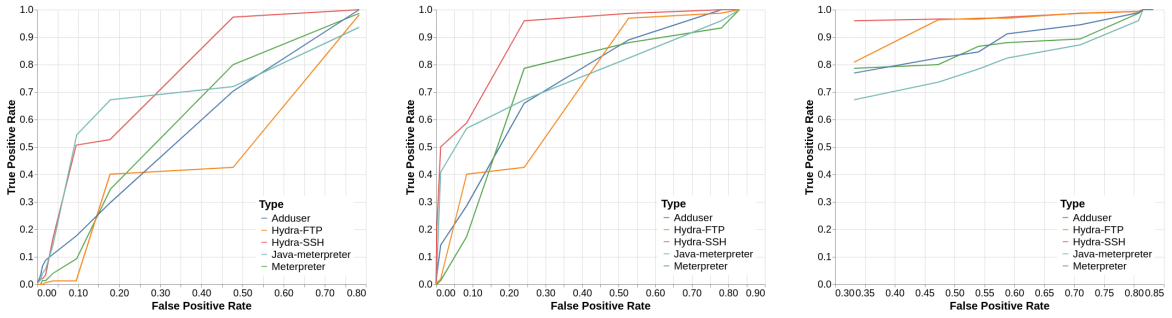
$$\tilde{w}_s(t) = \frac{\hat{w}_s(t)}{\sqrt{\sum_{t \in \mathbf{T}} \bar{w}_s^2(t)}} \tag{3.3}$$

The k-nearest-neighbor approach used by Miao Xie and Jiankun Hu classifies each data instance not by the respective label of the neighbors but by their distance. To this end, a radius is utilized (which is varied during the evaluation) that determines which neighboring data instances are taken into consideration for the labeling process. The actual label of the data instance is established based on the number of neighbors within the radius. If this number



(a) Count (b) IDF (c) Standardized IDF

Figure 6: *Classification results achieved by Miao Xie and Jiankun Hu [XH13]. Results are shown as an ROC curve for varying radii from* 0.1 *to* 1.0 *(step size of* 0.1*).*



(a) Count (b) IDF (c) Standardized IDF

Figure 7: *Classification results achieved by us. Results are shown as an ROC curve for varying radii from* 0.1 *to* 1.0 *(step size of* 0.1*).*

surpasses a user-defined threshold k (in the original paper 20), the instance is considered normal. Thus, their approach is a fundamental density-based anomaly detection algorithm.

Xie et al. evaluate the performance of their classifier by the ROC curves of the respective classes. The ROC (receiver operating characteristic) curve displays the relation between true positive and false positive rates depending on the classification confidence. The classification confidence is typically given in the form of the class probabilities, which corresponds to the radius for the here employed anomaly detection method. The area under the curve (AUC) is of special interest as this provides a good metric of the algorithm's classification performance. As the ROC curve assesses the tradeoff between true positive and false positive rate intuitively, the AUC indicates how well a classifier is able to assign a data instance of class 'X' to precisely

class 'X'. A perfect classifier has an AUC of 1 and a random classifier an AUC of 0.5. A value between 0 and 0.5 implies an inverse relationship between the prediction and the actual label.

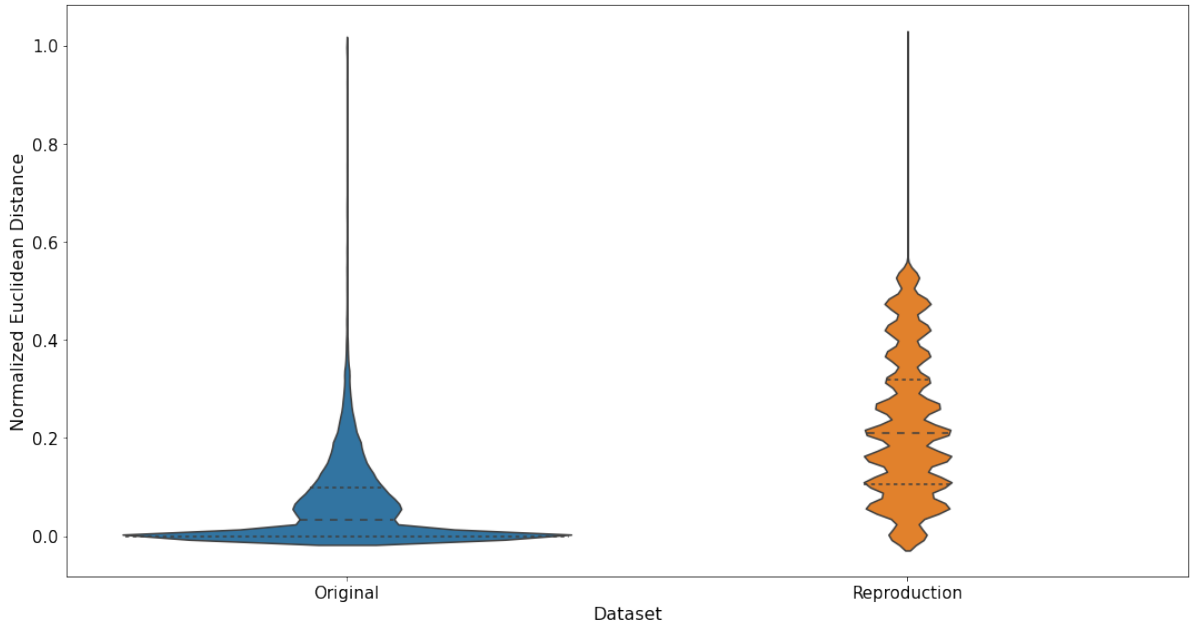


Figure 8: *Violin plot showing the normalized Euclidean distance to the* 20 *nearest neighbors (benign instances only) for both the original and reproduced ADFA-LD dataset (count frequency vectors).*

The performance of the classifier in the original paper is given in Figure 6, while Figure 7 showcases the results we achieve on our replica dataset. It is evident that we are not able to match the performance achieved in the original paper. However, there are quite a few interesting observations we made during the evaluation. First, we have to note that our classification is achieved by employing a threshold of $k = 100$ in comparison to the $k = 20$ in the original paper. This was necessary as the original value led to no meaningful classification of our data. This threshold of 100, while optimized to the extent possible for this work, might very well not yet be optimal. Nonetheless, it shows that there is a significant disparity between the structure of our reproduction and the original work. Primarily we can deduct a more evenly distribution of the benign data. Most likely, this effect is caused by the differences in benign data generation, which appears to be more diverse in our experimental setup. Another intriguing characteristic is the lower false positive rate observable in our visualizations. This once again implies a more even distribution of the benign data points in our set, consequently resulting in more benign instances still featuring more than 100 other instances with a distance of 0.1. Accordingly, the radius appears also to not be optimal for our dataset.

This difference can be visualized using a violin plot, as given in Figure 8. This figure shows the normalized euclidean distance to the nearest 20 neighbors for all benign data instances. Clearly visible is the stark difference in normal user behavior. While our data features a broad spectrum of activities resulting in a variety of clearly defined clusters with less apparent anomalies, the original dataset shows a single primary cluster with the majority of benign instances being in close proximity to each other, thus featuring a more defined set of outliers (resulting in the higher false positive rate). Conspicuous is the close to equal size of the clusters. This can be attributed to our benign data generation approach, which features a set of predefined distinct actions repeated over time. All in all, this again shows substantial inadequacies in the description of benign data generation. Regardless, we can still consider our goal of recreating a system call trace based dataset using the introduced toolset as being met while also clearly showcasing why dataset reproducibility is both essential and, unfortunately, for many datasets lacking.

3.2 On the Recreation of CICIDS2017

Table 5: *Class distribution of data instances.*

Class	Sampled CICIDS2017	Reproduction	Attack Category
normal	45461	45461	Normal
DoS Hulk	23107	30323	DoS
DDoS	12802	22186	DoS
DoS GoldenEye	10293	3835	DoS
DoS slowloris	5796	441	DoS
DoS Slowhttptest	5499	561	DoS
Heartbleed	11	162	DoS
Bot	1966	1966	Bot
Web Attack Sql Injection	21	92	Web Attack
Web Attack XSS	652	331	Web Attack
Web Attack Brute Force	1507	709	Web Attack
PortScan	15893	10104	PortScan
Infiltration	36	36	Infiltration
FTP-Patator	7938	10007	Brute Force
SSH-Patator	5897	5335	Brute Force

The CICIDS2017 (NIDS dataset of the Canadian Institute for cybersecurity) dataset is a modern network intrusion detection dataset containing network flows collected over a period of five days. They put a special focus on realistic background traffic as well as a broad selection

of up-to-date attacks [SHG18]. To evaluate our reproduction of the dataset, we compare it with classification results achieved by a previous work of ours [Bun18]. To this end, we preprocess both our dataset as well as the reproduction. The original dataset has a high class imbalance. To address this, we take a sample resulting in a new smaller version of it. This sampled dataset contains every 50th "benign" instance as well as every tenth instance of the classes "DoS Hulk", "DDoS", and "PortScan". For any other class, all instances are copied to the sampled dataset. Table 5 shows the composition of this sampled CICIDS dataset. We aim to preserve the category distribution between the two datasets. On the other hand, the attack distribution varies to a notable degree between both sets. This can be attributed to differences in the attack setup. Unfortunately, while providing excellent documentation of the laboratory setup, CICIDS's attack description is not as clear as required for perfect reproduction. However, we do not expect this to cause any issues, as our goal here is — as mentioned earlier — the evaluation of the versatility of the toolset and not the exact reproduction of the dataset.

In this context, it is also important to mention that we did not attempt to replicate the benign data instances. Instead, we relied on closely matching the experiment setup (both in hardware and software) and reusing the benign instances from the original CICIDS dataset. To generate the CICIDS dataset's benign user behavior, Sharafaldin et al. employ a so-called B-profile system, abstracting real captured user behavior to derive b-profiles [Sha+17]. These can then be utilized to create synthetic but very realistic benign traffic. Unfortunately, these profiles are not available to us, and thus, we are not able to recreate their benign data instances. For data protection reasons, we can not simply collect benign traffic. Accordingly, we would need to create our own approach to generating realistic synthetic benign background traffic. However, this would go beyond the scope of this work, as our main goal is the utilization of this toolset in the context of intrusion detection for homogeneous IoT devices. This holds especially true since it would give little advantage in the context of the primary goals of the tool evaluation. Thus, in the time that we could dedicate to the creation of such an approach, we would only be able to achieve a result subpar to the one reached by the researchers during the creation of the CICIDS dataset.

3.2.1 Experiment Setup

We attempt to match the original experiment setup as close as possible, given the resources at our disposal. The different machines are virtualized employing KVM with an individual network interface card passed through to every guest. We match the exact operating systems

utilized with the exception of MAC-OS[1], which we substituted with an Ubuntu 20.04 machine. Furthermore, we deviated with regard to the IPs assigned to the machines as we intend only to utilize the dataset stripped of all identifying features. Consequently, our victim network is composed of:

- Ubuntu Server 16.04 (running Damn Vulnerable Web-Application)
- Ubuntu Server 12.04 (with Heartbleed vulnerable OpenSSL Version and Apache2 web-server)
- Ubuntu Desktop 14.04 32-bit
- Ubuntu Desktop 14.04 64-bit
- Ubuntu Desktop 16.04 32-bit
- Ubuntu Desktop 16.04 64-bit
- Windows Vista SP2 32-bit
- Windows 7 Pro SP1 64-bit
- Windows 8.1 64-bit
- Windows 10 21H2 19044 32-bit
- Windows 10 21H2 19044 64-bit

Our attacker utilizes an up-to-date Kali (2021.4) installation as well as a Windows 10 64bit machine. All Linux machines are assigned three cores, 8196MB of memory, and 100GB persistent storage. All Windows machines receive an additional core.

3.2.2 Classification results

To evaluate the quality of the reproduced dataset, we compare classification performance with results obtained in our previous work "Anomalieerkennung mit Hilfe Neuronaler Netze" [Bun18]. In this work, we employed two types of neural networks combined with various preprocessing methods. For the purpose of this evaluation, we focus on the best results achieved by a **multilayer perceptron** (MLP) and a **recurrent neural network** (RNN). The MLP features a simple three-layer architecture with a total of 50 hidden units and is based on the paper "Using Artificial Neural Network in Intrusion Detection Systems to Computer Networks" [Dia+17]. On the other hand, the RNN is based on "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks" [Yin+17]. It features ten time

[1] A fitting MAC-OS system was not available to us.

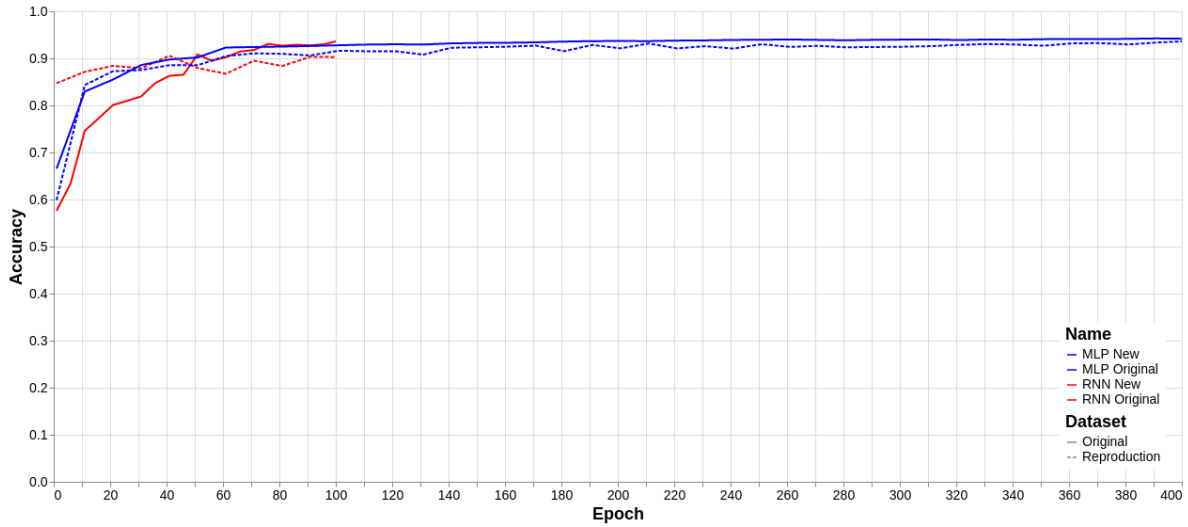
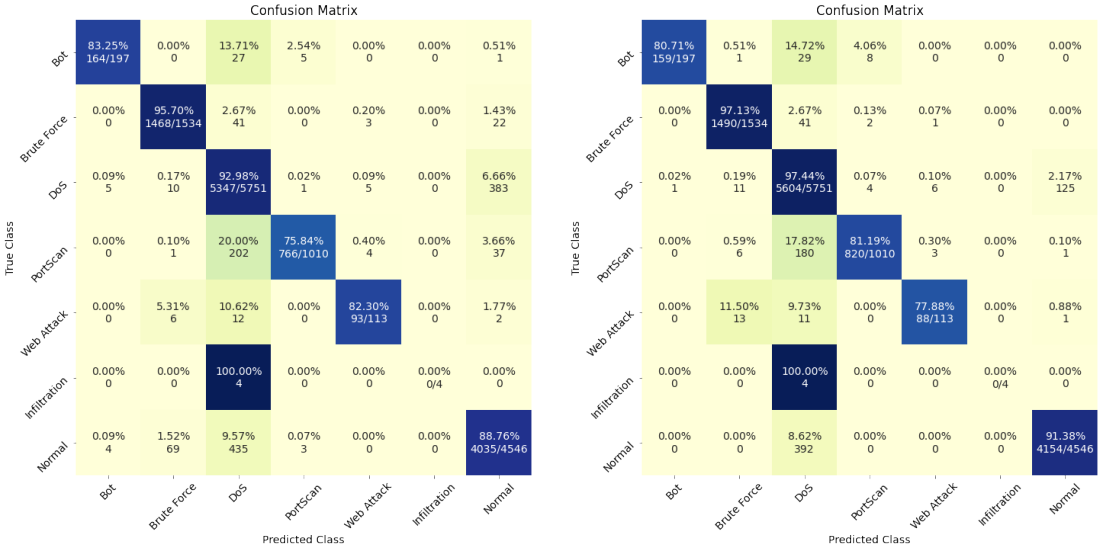


Figure 9: *Accuracy in relation to epoch, showing both the original and our reproduction dataset.*

steps and 80 hidden units arranged in a single hidden layer. Regarding preprocessing, we rely on the results obtained by our previous work and choose standardization for the RNN and logarithmic scaling in combination with min-max scaling for the MLP (for a description of the preprocessing methods see section 5.6.1) [Bun18]. Additionally, all constant features are removed from both datasets. Important to note is that these are eight for the original dataset, but only five for the reproduced, with the difference being the features "Bwd Avg Bytes/Bulk", "Bwd Avg Packets/Bulk", and "Bwd Avg Bulk Rate". However, an exclusion of these three features in the reproduction does not lead to a significant difference in classification accuracy (max. Δ is with 0.0028 within the expected margin of error).

Figure 9 shows the accuracy of both models with respect to epoch and dataset. The maximum accuracy on the original datasets employing an RNN is 93.62% while we only achieve an accuracy of 90.05% for the reproduction datasets. In the case of the MLP, we achieve an accuracy of 94.23% and 93.61% for the original and reproduced dataset, respectively. This is within the margin of error we expect based on the difference in class distribution as well as the preprocessing being optimized for the original dataset. In particular, it should be noted that we observed accuracy deltas of $\sim 30\%$ depending on preprocessing choices during our previous work on the CICIDS2017 dataset [Bun18].

To evaluate the impact of the adoption of the benign data instances, we can consider the confusion matrices attained on the reproduced dataset (cf. Figure 10). It is clearly visible that a considerable amount of mislabeled data instances are either normal but mislabeled as

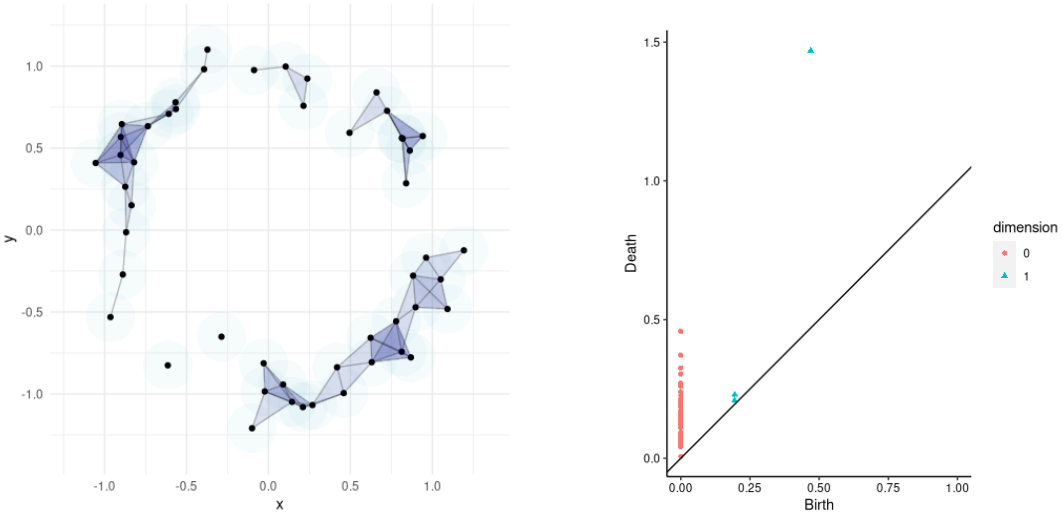


(a) RNN (b) MLP

Figure 10: *Confusion matrices CICIDS reproduction dataset.*

a malicious class or the opposite. The majority of these false positives and negatives belong to the class "DoS". This is a behavior that we also observed while working with the original dataset [Bun18]. Accordingly, our experiment setup mirrors the original one close enough such that the retention of the original benign instances is unproblematic. Thus, the reproduction of the CICIDS dataset can be considered a success.

4 Topological Data Analysis



(a) Vietoris Rips Complex

(b) Persistence Diagram

Figure 11: *Left: Vietoris Rips complex with filtration distance $\epsilon = 0.15$. Right: Persistence diagram showing the persistent homology of the dataset given in Figure 11(a)*

In the following, we present a coarse overview of the concept of topological data analysis and persistent homology in particular. As we are limited by the scope of this thesis, we only provide an intuitive introduction to the topic while primarily focusing on introducing the feature extraction and classifiers employed during the classification. For a more in-depth discussion, we recommend the book "Data Science for Mathematicians" by Nathan Carter as an excellent introduction to this field [Car21].

Topological data analysis (TDA) analyses the shape from which a finite set of data instances is sampled by utilizing topological features (properties). These topological features express the connectivity of the given dataset through the quantification of i-dimensional holes [Moo+20]. The respective data is approximated by a data structure called simplicial complex [Ott+15]. As the name implies, a simplicial complex is a collection of simplices. Each simplex is a simple geometric structure whose shape depends on its dimension. A zero-dimensional simplex is a

point, one-dimensional a line, two-dimensional a triangle, three-dimensional a tetrahedron, and so forth. A simplicial complex thus defines a triangulation of a given topological space [Car21].

```
   input  : ϵ, dataset
   output : simplicial complex
1  foreach point in dataset do
2  |  Draw sphere centered at point with radius ϵ;
3  end
4  foreach dimension n = 2,3,4,... do
5  |  foreach set of n+1 points whose spheres intersect do
6  |  |  Draw n-dimensional face between the n+1 points;
7  |  end
8  end
```

Figure 12: *Construction of a Čech complex.*

There are multiple ways to compute a simplicial complex of a given data set. One of the most well-known and intuitive ones is the Čech complex. Its simplified computation is given in Figure 12. Intuitively, the filtration distance ϵ defines a radius of a hypersphere centered at each data point. If $n+1$ sphere overlap, we establish a n dimensional simplex. Due to its high computational cost, we instead rely on an approximation of the Čech complex, the Vietoris Rips complex. This variant trades accuracy for greater computational efficiency by solely utilizing the distance between pairs of data points instead of the intersection [Ott+15].

Figure 11(a) shows the Vietoris Rips complex of a set of data instances in two dimensions sampled from a circle. Intuitively the main characterizing property of the underlying shape is its hole. Topological features capture allow us to capture this underlying structure with high resilience to noise. This is due to the fact that topological properties are invariant to homeomorphisms (i.e., they allow stretching and bending of the given space) [Car21]. Consequently, employing TDA, we can derive a robust description of the nature of the dataset, which has many benefits in fields like anomaly detection.

In the following, we introduce a key concept of TDA, persistent homology, (cf. section 4.1), before discussing how it can be utilized in the context of machine learning and anomaly detection as a means of feature extraction (cf. section 4.2). Later in this thesis, we examine more involved methods to incorporate the concept of persistent homology directly into the classification and anomaly detection process (cf. section 4.3).

4.1 Persistent Homology

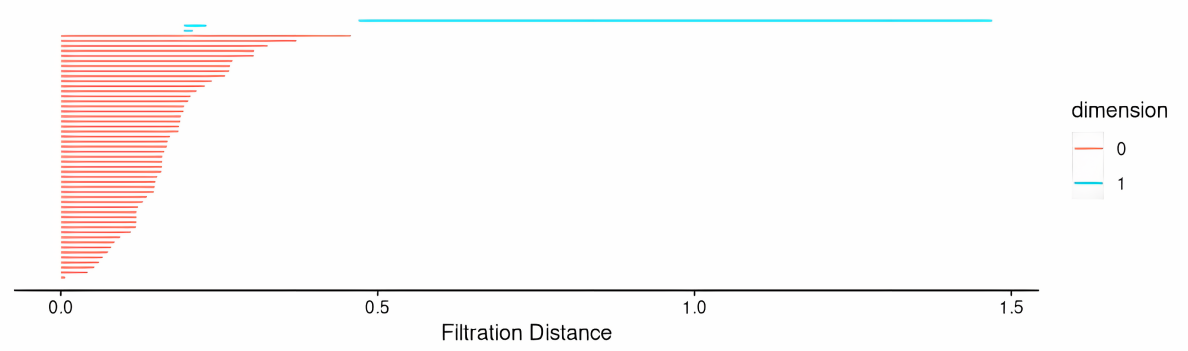


Figure 13: *Barcode showing the persistent homology of the dataset given in Figure 11(a).*

Persistent homology (PH) facilitates the description of the structural properties of a given data set [ZJZ21]. PH considers the forming and disappearance of i-dimensional holes (topological features) within the simplicial complex as we grow ϵ (filtration distance) [Car21]. In its simplest case — observing only 0-dimensional features (i.e., connected components / clusters) — it thus behaves very similarly to hierarchical agglomerative clustering. This means it allows us to observe and capture the forming and merging of clusters, starting from every point being its own cluster until only one cluster is left. The same concept applies to higher-dimensional holes. In other words, persistent homology is not only capable of giving us the number of holes in a simplicial complex but tracking holes over an iteration of simplicial complex generated by varying ϵ. **Thus, a data structure is characterized by the lifetime of the individual holes.** There are two typical representations: A barcode where each lifetime is expressed as a bar with the x-axis showing the filtration value (cf. Figure 13), and a persistence diagram where the x-axis is the birth- and the y-axis the deathtime of each hole (cf. Figure 11(b)). In the following, we give an overview of persistent homology-based feature extraction in general and then discuss the three methods evaluated in this thesis as well as the two classifiers.

4.2 Persistent Homology based Feature Extraction

All three feature extraction methods utilize the birth and death time of the holes in the underlying data to generate the feature embedding. To this end, we consider the respective persistence diagrams, which are computed from set of sensor data from all devices over n time steps. This section introduces the three methods and provides reasoning behind their selection.

4.2.1 Feature Extraction employing Defining Quantities of Persistence Diagrams

Feature extraction by defining quantities is the simplest of the three methods. Cang et al. introduce this approach in "A topological approach to protein classification" [Can+15]. All following refers to their work. In essence, the persistence diagrams in each dimension are represented by standard statistical quantities. For each dimension, we consider the persistences' mean, median, standard deviation, sum, minimum, the persistence of the three longest lifetimes, and the birth time of the longest lifetime.

The reasoning behind the choice of this extraction method is its simplicity. It is the easiest way to generate a numeric vector from a persistence diagram and thus serves as a baseline against which the other methods can be compared. Additionally, due to its simplistic nature, it is also computationally inexpensive.

Persistence Entropy

A sub-category of this approach is the representation of the barcodes by their entropy. This approach is presented in "An entropy-based persistence barcode" by Gonzalez-Díaz et al. [Chi+15]. For each homology dimension, the Shannon entropy is calculated based on the normalized lifetime of its topological features. The entropy for a dimension given normalized lifetime p_j with $j \in J$ corresponding to the individual barcodes is thus defined as:

$$H = -\sum_{j \in J} p_j * \log(p_j) \tag{4.1}$$

The lifetimes are normalized by the sum of all lifetimes for the respective homology dimension.

4.2.2 Feature Extraction employing Binning of the Persistence Diagrams

Binning denotes the process of discretizing a continuous range into equal-sized bins. In this case, the continuous range is the filtration distance. This approach is adapted from the paper "Integration of element specific persistent homology and machine learning for protein-ligand binding affinity prediction" by Wei et al. [CW17]. To this end, we assign all points whose birth time is before and its deathtime after the start of the bin to that bin. Thus, a given persistence diagram is expressed as a vector, where each entry represents one bin and its value the number of persistence diagram points contained by this bin. This process is repeated for each dimension and the resulting vectors are ultimately concatenated.

While still a simple approach, binning allows capturing the persistence in greater detail (depending on the number of bins). Thus, as a method, it falls between feature extraction by defining quantities and more involved methods like the distance-based approach, which we discuss in the following section.

4.2.3 Feature Extraction employing Pairwise Distance between Persistence Diagrams

The distance-based feature extraction considers the pairwise distance between all persistence diagrams in each respective dimension to generate the embedding. It is based on "Bot Detection on Social Networks Using Persistent Homology" [NAA20]. They utilize multidimensional scaling to generate a lower dimensional embedding from the aforementioned distance matrix.

Throughout the evaluation, we vary the distance metric, with the two employed metrics being Bottleneck and Wasserstein. These are the de facto standard distance metrics employed to calculate the similarity between two persistence diagrams. Essentially, both methods align the persistence diagrams (here $\mathcal{D}$ and $\mathcal{E}$) with the Bottleneck distance being the maximum distance between two points and Wasserstein distance being the pth root of the sum of all distances to the power of p (cf. equations 2.1/2) [Hof+17].

$$w_\infty(\mathcal{D},\mathcal{E}) = \inf_\eta \sup_{x\in\mathcal{D}} \|x-\eta(x)\|_\infty \tag{4.2}$$

$$w_p^q(\mathcal{D},\mathcal{E}) = \inf_\eta \Big(\sum_{x\in\mathcal{D}} \|x-\eta(x)\|_q^p\Big)^{\frac{1}{p}} \tag{4.3}$$

For our purposes, we choose $p = 1$ and $q = \infty$ known as the 1-Wasserstein distance.

Distance-based feature extraction allows us to capture characteristics that are common for different clusters within the respective classes. This gives us a finer granularity with which we can differentiate between normal and abnormal behavior patterns. Thus, this approach should be able to capture relevant information that both defining quantities and binning fail to.

4.2.4 Feature Extraction employing Codebooks

The approach of utilizing codebooks to extract features from the persistent homology of a given point cloud is discussed by Zielinski et al. [ZJZ21]. The codebooks are established following a bag-of-words approach with a soft assignment. To this end, the points in a sample

of the persistence diagrams are consolidated into a single diagram. The sample is weighted to increase the influence of topological features with longer lifetimes, as these typically carry more information. Weights are established for a given lifetime t following:

$$w_{a,b}(t) = \begin{cases} 0 & \text{if } t < a \\ \frac{t-a}{b-a} & \text{if } a \leq t < b \\ 1 & \text{if } b \leq t \end{cases} \tag{4.4}$$

where a and b are the lower and upper 5% quantile of the lifetime, respectively. This collection of points, each representing a topological feature's respective lifetime, are clustered by employing a Gaussian mixture model. The resulting Gaussians are the codewords. For every data point per persistence diagram, we can now compute the likelihood with which it was generated for all Gaussians, respectively. These are then summed and weighted by the individual Gaussians weights to generate the feature vector. For proof of the stability of this approach, refer to the reference paper.

We expect this method to perform best for our specific use case. The weighted sampling in combination with soft assignment should result in a robust characterization of normal behavior. Consequently, abnormal behavior both with respect to the device's own past but also the other homogeneous devices should be better highlighted than in the other methods.

4.3 Persistent Homology based Classification

We employ two classification methods that directly incorporate persistent homology into the classification process. These are TDA-based multiclass classifiers presented in "Classification Based on Topological Data Analysis" by Kindelan et al. and an autoencoder integrating persistent homology "Topological Autoencoder" by Moor et al. [Kin+21; Moo+20]. The respective algorithms are chosen as they represent a method purely relying on topological data analysis and, on the other hand, one that uses TDA to augment and improve the classification process. Furthermore, both are focused on anomaly detection with good performances in their respective reference papers. Thus, we hope to achieve an improved classification performance in relation to more traditional approaches in our use cases (which rely heavily on detecting abnormal behavior). In the following, we first introduce the exclusively TDA-based classifier and then follow up with a description of the autoencoder.

4.3.1 TDA-based Classification Algorithm

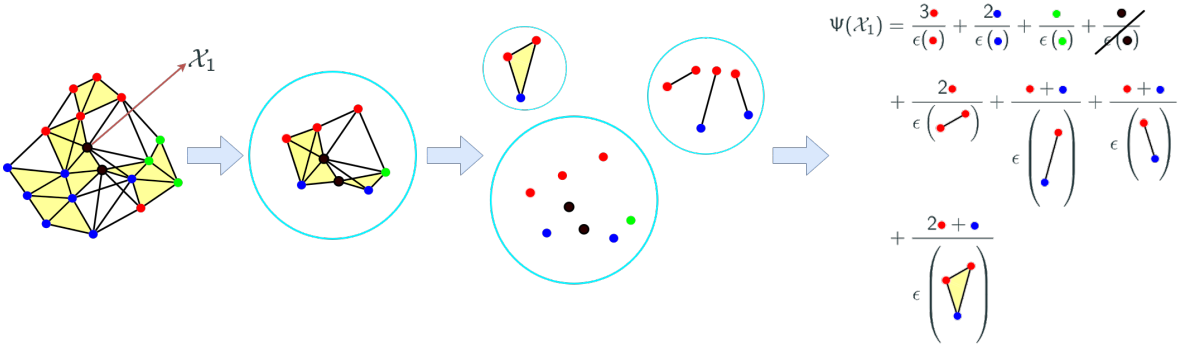


Figure 14: *Visualization of TDABC's classification process.*

This section gives an overview of the functionality of the TDA-based classification algorithm (TDABC) by Kindelan et al. As such, all following refers to their work [Kin+21]. The TDABC is a nearest neighbor based classifier. However, instead of all neighbors within a certain radius (or a fixed number of closest neighbors), the algorithm utilizes the closed star of any given point in the simplicial complex given a certain filtration of the space. The star of a simplex (in our case a 0-simplex) is defined as the set of all cofaces of this simplex in the simplicial complex [Kin+21]. Here all simplices intersecting with an infinitely small hypersphere around a 0-simplex are its cofaces. Additionally, we consider the closure of the star, so all 0-simplices that form the respective higher-dimensional ones. The label is determined based on the points that form the star weighted by filtration distance and respective i dimensional simplices. Figure 14 exemplifies this process. Given is a simplicial complex, with the black points representing unlabeled test data and X_1 pointing out the instance to be labeled. The respective simplices are colored in red, blue, and green depending on the class they belong to. We now consider the closed star of the marked 0-simplex and the algorithm breaks down the complex into its i dimensional simplices. These now cast a weighted vote for the label of the instance. This leaves two main questions, which we discuss in the following:

1. How is the filtration distance ϵ selected?
2. How are the points weighted?

Filtration distance selection

For the filtration distance selection, persistent homology comes into play. Specifically, after the calculation of the persistent homology, a sub complex is selected using the birth time of a cer-

tain persistence interval with a dimension of at least one. The reference paper discusses three methods for choosing such an interval, the simplest being selection at **random**. Alternatively, the longest interval (**max**) or the interval whose length closest matches the average lifetime (**mean**) can be selected. The respective birth time then determines the filtration distance.

Label Weighting

Given the star of the respective point, we now consider each i-simplex starting with dimension 0. Each data instance casts a vote for its own label, which is weighted by $1/\epsilon$ where ϵ is the filtration distance at which the simplex became part of the connected component. We repeat this process for higher-dimensional simplices, with every point's label within the simplex being weighted by the filtration that births the simplex. After all, votes are cast, we have a map of labels to confidence values and simply choose the label with the highest confidence as our prediction. In the case of a tie, we draw from the set of tied labels. If no labeled point is part of the star, the point is labeled at random.

Use case relevant Performance

The reference work compares the algorithm's performance with one of the classic nearest neighbor approaches on a collection of artificial and real-world datasets. In particular k-nearest neighbors ($k = 15$) and its weighted variation (wk-nearest neighbors) are employed. While the proposed method achieves excellent results on almost all of the datasets, especially important for us is its performance on the wine dataset, as it closely matches the expected distribution of the classes in our datasets. As the performance metric, we consider the Matthews Correlation Coefficient (MCC) as it is, in our opinion, the best single metric to evaluate model performance, regardless of class imbalance. It is defined as:

$$\frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{4.5}$$

where TP, TN, FP, and FN correspond to true positives, true negatives, false positives, and false negatives. MCC measures the correlation between the prediction and actual label and falls into the interval $[-1, 1]$ where -1 corresponds to perfect negative correlation, 0 to no correlation, and 1 to perfect positive correlation. Given a maximum simplicial complex dimension of six and using the average lifetime as a selector, TDABC achieves an MCC of 0.514, which is respectable given the amount of overlap within the dataset. On the other hand, k-NN and wk-NN achieve an MCC of 0.400 and 0.465, respectively. However, while

outperforming or at least matching the classic nearest neighbor approaches, TDABC struggles on the Swissroll dataset. The dataset is generated with a high amount of noise, resulting in regions with a significant overlap of all present classes. While we can also observe class overlap in the Wine dataset, they are limited to two of the classes at a time. Nevertheless, although comparatively worse with an MCC of 0.752, in contrast to the 0.815 of wk-NN, the results are still serviceable.

4.3.2 Topological Autoencoder

Moor et al. introduce topological data analysis to the concept of autoencoders by evaluating the persistent homology of the respective mini-batches in latent and input space [Moo+20]. All following refers to their work [Moo+20]. The concept proposed by Moor et al. is fundamentally a regularizer based on the change of topology of individual mini-batches before and after encoding. Consequently, it is independent of the network architecture. As a result, we utilize their approach to augment an autoencoder already part of our classification pipeline. The autoencoder (TDAENC) features five layers per encoder and decoder, respectively, with leaky ReLU as an activation function. In the following, we detail the calculation of the topological loss.

Topological Loss

The key idea is the regularization of the reconstruction loss based on the difference between the persistent homology of each mini-batch in input and latent space to preserve topological structures in the encoding. The first step is the computation of the persistent homology of the Vietoris Rips complex of the pairwise Euclidean distance between the data instances in the mini-batch (both in latent and input space). Hereby, the original paper — based on experimental research — only focuses on zero-dimensional topological features. We now consider the simplices birthing and destroying the zero-dimensional topological features. Specifically, we focus on edges as they are the destroying simplex for zero-dimensional topological features. This edge is then mapped to the Euclidean distance between the two vertices establishing it. The regularization term is calculated based on the difference in distance between the vertices in latent and input space. In detail the regularization ($\mathcal{R}_t$) for latent space $\mathcal{Z}$ and input space $\mathcal{X}$ is calculated as follows:

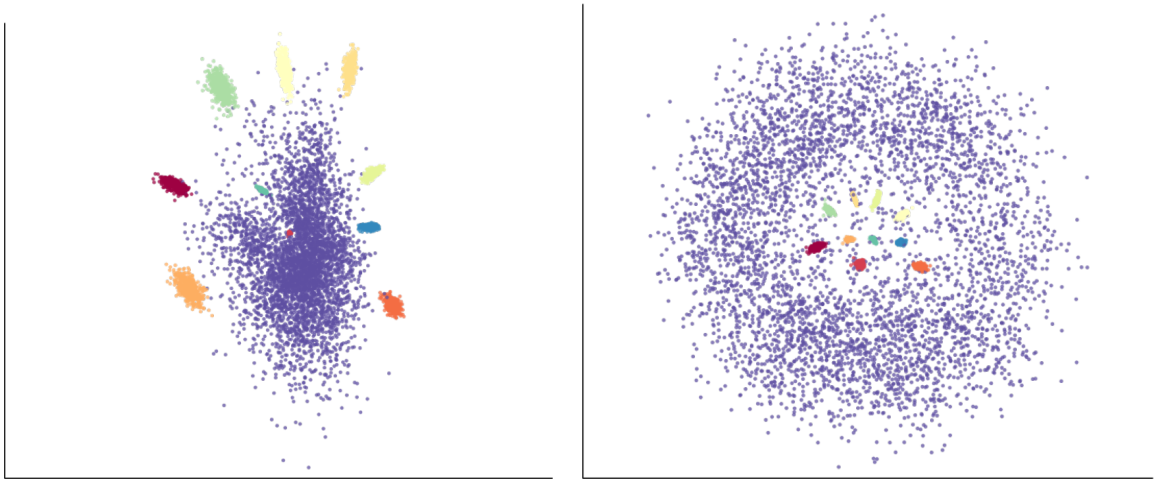
$$\mathcal{R}_t = \mathcal{R}_{\mathcal{X}\to\mathcal{Z}} + \mathcal{R}_{\mathcal{Z}\to\mathcal{X}} \tag{4.6}$$

where

$$\mathcal{R}_{\mathcal{X}\to\mathcal{Z}} = \frac{1}{2}\|D_X(p(X)) - D_Z(p(X))\|^2 \tag{4.7}$$

$$\mathcal{R}_{\mathcal{Z}\to\mathcal{X}} = \frac{1}{2}\|D_X(p(Z)) - D_Z(p(Z))\|^2 \tag{4.8}$$

Here X and Z are the mini-batch in input/latent space, D is the Euclidean distance matrix for the respective batches and p gives us the persistence pairings.



(a) Autoencoder (b) Topological regularized autoencoder

Figure 15: *Latent embeddings of a dataset containing points sampled from multiple spheres, enclosed by a larger sphere [Moo+20].*

Use case relevant Performance

To quantify the effect of the topological autoencoder in comparison to the same autoencoder but unregularized, the reference work considers the class layout within the latent space. Specifically, the latent representation of a sphere dataset is studied. This dataset consists of ten 100 dimensional spheres enclosed by an eleventh sphere, featuring as many points as the ten inner combined, all of them lying within a 101 dimensional features space. The respective latent representations are given in Figure 15. It is clearly visible that the autoencoder on its own is incapable of capturing the structure of the dataset (cf. Figure 15(a)). On the other hand, its topological variant not only captures the structure of the data but still provides adequate

separation of the classes in the lower dimensional space (cf. Figure 15(b)). Consequently, we hope for better separation based on more minute details within our own datasets.

5 Experiment Setup

The goal of the experiments is to ascertain the feasibility of topological data analysis based anomaly detection for homogeneous IoT devices. To this end, we employ two experiment setups inspired by real-world scenarios, namely a centralized and a distributed usage scenario. The most important criterion is the role of an IDS in the respective scenarios. While this is straightforward in the centralized case (i.e., central server for data collection and IDS), for the distributed network, multiple options exist. We choose self-contained intrusion detection, where the devices decide based on a majority vote if any of them is behaving anomalously. This is mirrored in the data collection process by broadcasting the sensor data to all IoT devices in the network.

As IoT devices, we employ Raspberry Pis. The actual usage of the devices in the scenarios does not have to be perfectly emulated, as the primary objective is the generation of realistic background noise. As such, we rely on a camera extension module that allows the Pis to be turned into IP cameras. This provides realistic noise for both scenarios. We prefer to use Raspberry Pis instead of proprietary devices due to their ease of accessibility and extensibility.

In the following, we detail the experiment setups as well as our reasoning behind the choices we made. We first introduce the setup for both the centralized and distributed case (cf. section 5.1 and 5.2 respectively). Afterward, we discuss our attack selection and motivate our configuration and application in section 5.3. Following, we give details on the composition of the resulting datasets (cf. section 5.5) and introduce the pipeline employed for the classification and feature extraction procedures, given in section 5.4. Finally, we specify the classification procedure utilized during the evaluation 5.6.

5.1 Centralized Data Collection

The first scenario is simplified modeling of a security camera setup that — for example — could be found at an airport. Without loss of generality, we can make a number of assumptions. It is reasonable to assume that this set of devices is homogeneous. However, the device links

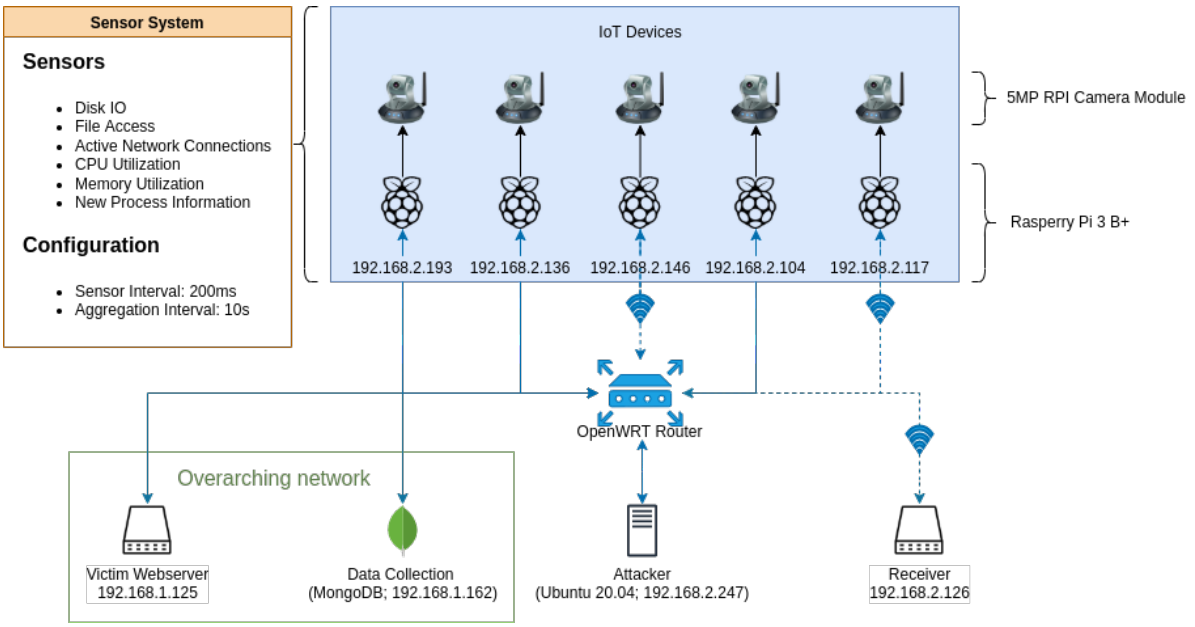


Figure 16: *Setup centralized data collection.*

might be disparate, featuring wired and wireless connections. Furthermore, the cameras and their firmware are from a third-party vendor, who also manages security updates. The airport security is equipped with a central computing unit that is utilized, for example, to store certain recordings of the cameras but can also be employed to host a centralized IDS such as T-ChangeDrift. The homogeneous set of cameras is part of a larger heterogeneous network of (non) IoT devices, including the likes of routers, scanners, and the like. Additionally, the OS of the security cameras is Linux-based.

5.1.1 Device Setup

Figure 16 details the exact experiment setup for the first scenario. As mentioned earlier, we utilize Raspberry Pis as IoT devices. The exact model is a Raspberry Pi version 3 B+ with a 1.4GHz quad-core CPU and 1GB of memory. Each Pi runs up to date 32bit Raspbian built upon Debian 11 with the Linux kernel version 5.10. Two of the devices are connected wirelessly (wireless standard 802.11 a/c, 2.4GHz range), while the rest are connected via Ethernet. Accordingly, a discrepancy exists between the wireless and wired devices with regard to network speed. However, we consciously opt for this deviation from complete homogeneity as such a difference is to be expected in real-world scenarios. Each Pi is outfitted with a camera module using its CSI camera connector. The camera module in use employs a

5MP sensor and has the model name "RPI-CAM-A". We utilize this camera to stream a 480p livestream to a separate machine (denoted as "Receiver" in the graphic) via "raspivid" using the script given in Listing 5.1.

```
#!/bin/bash
raspivid -o - -t 0 -w 640 -h 480 -fps 25 -b 1500000 -rot 180 | cvlc -vvv stream
    :///dev/stdin --sout '#standard{access=http,mux=ts,dst=:8090}' :demux=h264
```

Listing 5.1: *Video streaming script.*

Besides the IoT devices, we require a number of additional hardware. First and foremost, this includes the router. Here we use a TP-Link Archer C6 running OpenWRT version 19.07. Additionally, we need a central data collection server. In this case, this is a Docker container with MongoDB version 5.0.6 hosted on a server outside of the subnetwork our Pis are located in. Another server located outside the original network is a webserver that takes the role of a victim for a subset of the attacks. Lastly, there is the attacker's hardware. We utilize a dedicated machine with an Intel i5 4200U, 8GB of memory, and Ubuntu 20.04 as the operating system.

5.1.2 Setup Validity

To motivate the validity of this experiment setup, we now consider several different factors and argue why our choices in the given context are justified. In particular, these are:

1. Role of the attacker
2. Setup of IoT devices
3. IDS/Data collection
4. Network structure
5. Other infrastructure

As detailed in the first paragraph of this section, we can expect the network of IoT devices to be part of a more extensive network of heterogeneous devices. It is not unreasonable to assume a potential corruption of one of the machines, consequently giving validity to the **role of the attacker** in our network setup. The **setup of IoT devices** concerns the choice of device, task, and operating system as well as the number of devices and network setup. The majority of these are directly motivated by the usage scenario. Specifically, we consider a scenario employing IP cameras. As such, we can assume a Linux operating system and the

devices — or at least a large enough subset of the devices — to be part of the same network. As argued earlier, the choice of Raspberry Pis as hardware is justified due to their ease of use and the fact that we primarily are concerned with all devices being homogeneous. This leaves the number of devices. Here it is essential to find a balance between a setup capable of generating realistic data while providing enough devices to perform anomaly detection and outside factors limiting the number (e.g., financial, data collection/preparation) on the other side. Therefore, we aim to minimize the number of devices without compromising the desired use case. In our opinion, this is given with five devices, as a 20% infection rate — while high — is not unreasonable, and this number of devices is able to generate sufficient data in the peer-to-peer scenario, especially considering that in a real-world scenario multicast instead of broadcast or separation into local groups is more likely [Kha+19].

All data we collect is observed from the point of view of the IoT devices. Thus, there is no quantifiable difference between the **data collection** procedure and an IDS. This is due to the passive nature of an IDS, which acts merely as an observer from the standpoint of the devices. Consequently, we do not need to model an IDS service in addition to the data collection. The server's position is also justified as it is reasonable to assume that such a central server is part of an overarching network managing multiple devices in multiple different subnets. This also answers the question of the validity of the **network structure**.

We already discussed the data collection as well as the attacker; hence, the only **other infrastructure** left is the receiver as well as the victim server. The victim server acts as a placeholder for a server outside the network that — given an infection with an IoT botnet — might be attacked by the corrupted devices. As we intend to strip the destination IP from the dataset, its position within the same network as the data collection server is irrelevant. This information removal is necessary, as due to the nature of such an experiment, we have to limit the IP addresses associated with malicious behavior to a greater extent than observable in the real world. Thus, the addresses carry substantially more information about the state of the system than normally expected. As mitigation, we reduce the information to the differentiation between expected and unexpected connections. This information is encoded by the output of multiple sensors, and thus thanks to this redundancy, we can safely replace the destination IP with a replacement integer ID during the aggregation.

The receiver's sole purpose is to act as a destination for the video streams to provide realistic network traffic in the data collection. Thus, its exact hardware/software setup is irrelevant as the only requirement is to be sufficiently powerful to handle the five data streams. The only important criterion is that this server is reachable at the same address during the entire

experiment as everything else would be anomalous behavior not typically found in a real-world equivalent. Notable is also the wireless connection of the receiver. However, since it provided the necessary bandwidth for the five streams, it has no further impact on the dataset generation.

5.2 Distributed Data Collection

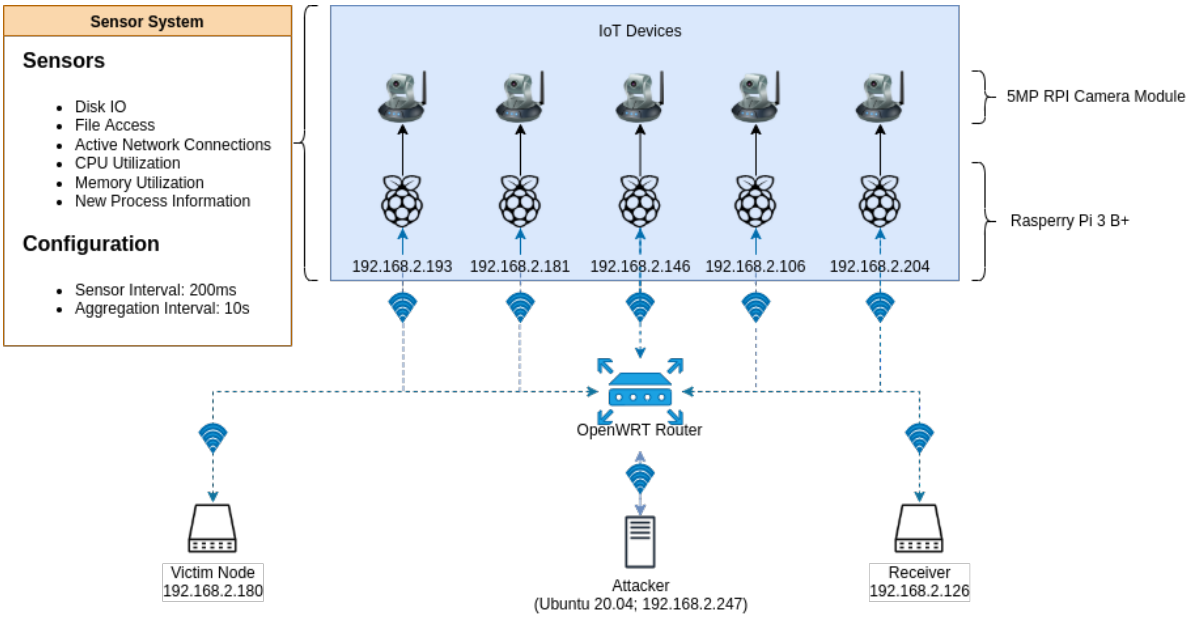


Figure 17: *Setup distributed data collection.*

The second scenario shares many similarities with the centralized one. As such, we focus here on the key differences. Figure 17 gives a schema of the experiment setup. We once again assume homogeneity of the devices, utilizing the same Raspberry Pis as a stand-in. In this scenario, we are considering the prototype of a mobile ad-hoc network. A concrete example is a vehicular ad-hoc network exchanging sensor information regarding road and traffic conditions. Accordingly, the device links are expected to be wireless. It is furthermore reasonable to assume that the devices themselves are Linux-based. In the following, we discuss the setup validity. Regarding device setup refer to section 5.1.1.

5.2.1 Setup Validity

The majority of the setup validity follows the argumentation given in section 5.1.2 for the first scenario. However, regarding the five factors, a subset is important to reconsider. Namely, these are the **role of the attacker**, the **setup of IoT devices**, and the **network structure**. It is reasonable to assume that a malicious or compromised node can join the network. Given the nature of an ad-hoc network, especially a vehicular one, a constant change of nodes and, in particular, the joining of previously unknown ones is to be expected. Thus, the **role of the attacker** as a node within the network is valid.

Regarding the **setup of IoT devices** and the **network structure**, we take some liberties to simplify data collection. We follow a realistic setup by utilizing wireless links but deviate by employing a central router. However, we argue that this change has no impact on the quality of data, given the lens through which the dataset observes the network, The device state can only be observed through the respective sensor data. Given our sensor selection, the presence of a dedicated router is not detectable but simplifies our data collection. Therefore, we argue that the network structure is valid. We make a similar argument about the device setup. Specifically, we once again utilize IP cameras. The device itself is not capable of observing what kind of payload is calculated (nor does it directly impact the attack detection). The only important factor is its realism. The payload has to feature randomly occurring anomalies that are subject to real-world occurrences observed by the device to achieve this. Our IP-camera setup is capable of generating such anomalies. The only problematic area is the streaming of the data. However, this is static and consistent behavior and thus can simply be filtered from the network data, therefore giving validity to the IoT devices' setup.

5.3 Attack Selection

We explore a total of 14 different attacks belonging to five different categories (cf. Table 6). The goal is a realistic and representative selection of attacks to showcase a broad area of typical attacks against IoT devices. To this end, we consider malware belonging to the families *botnet*, *cryptojacking*, *ransomware* as well as *(D)DoS* and various methods from the fields of *reconnaissance and infiltration*. In the following, we briefly discuss each category, the motivation behind its attack selection, and relevant decisions made during the attack process.

(Distributed) Denial of Service) ((D)DoS) attacks are some of the most frequent attacks observed [Ang17]. Their goal is to overwhelm a device to prevent it from performing its

Table 6: *Attack selection and respective categories.*

Category	Description	Attacks
Botnet	Hijacks device to perform attacker scheduled tasks (e.g., DoS attack against a specific domain).	Represented by the attacks: ▪ Ares ▪ Bashlite ▪ Mirai
Cryptojacking	Exploits compute power of infected host to mine cryptocurrency.	Represented by the attacks: ▪ Cryptominer
(D)DoS	Overloads resources of the victim to cause an interruption to its designated functionality.	Represented by the attacks: ▪ GoldenEye ▪ Hulk ▪ Slowloris
Infiltration & Reconnaissance	Comprises various attacks with the goals of discovering vulnerable machines, gaining access, and achieving persistence.	Represented by the attacks: ▪ Nmap ▪ Cloud Download ▪ SSH Patator ▪ FTP Patator ▪ Metasploit
Ransomware	Encrypts files on the host with the goal of extortion.	Represented by the attacks: ▪ Ransomware

intended functionality. Most commonly, either the network link or the processing capabilities (memory and computing) are overloaded [Ang17]. Due to the limited resources of IoT devices, they are easily incapacitated by such attacks [AK15]. Accordingly, integration of these attacks into the attack scenario is essential. We consider three attacks from this family, namely "GoldenEye", "Hulk", and "Slowloris". The selection of these three is due to them being readily available as well as their extensive use in other reference work [SHG18; CH13; Cat+21].

Cryptographic attacks, especially **ransomware**, have been on the rise for years at the present time. Just in 2020 alone, the number of ransomware attacks has risen by 62% [Bla21]. Thus, we consider it important to include these types of attacks. The inclusion of a **cryptojacking** might seem unintuitive at first, given the low compute power of IoT devices. However, other

factors benefit their usage, mainly the often subpar security and largely self-contained nature of the devices (i.e., a miner can run for long periods of time before getting noticed), while the large number of devices can mitigate the lack of computing power. Examples of these in the wild are specialized malware like "PyCryptoMiner" or "Rabbit" as well as adapted malware like "Mirai", with all targeting specifically IoT devices [Bra18]. Therefore, we see merit in including these types of attacks in the dataset. As ransomware, we employ a simple script that alternates between encrypting and decrypting all files in a given directory. To simulate a malware mining cryptocurrency, we use *solominer*, a simple Python script mining Bitcoin [ice21].

```
1 #!/bin/bash
2 nmap -sS -sU -T4 -A -v -PE -PP -PS80,443 -PA3389 -PU40125 -PY -g 53 -script "
      default or (discovery and safe)" 192.168.2.193
```

Listing 5.2: *Nmap script.*

As with other attacks, preceding the infection is **reconnaissance and infiltration**. Typically, IoT malware is self propagating [Wan+17]. This means it scans the network (internet) for other vulnerable devices and exploits the found vulnerability. To mimic the scanning, we utilize Nmap, which is the de facto standard network scanning utility in cybersecurity (cf. script 5.2). A large portion of IoT malware relies on weak authentication to exploit other devices [Wan+17]. Accordingly, we utilize two brute force tools that find a lot of usage in reference papers to simulate this behavior [SHG18]. In addition, we also group Metasploit into this category. While not a reconnaissance tool in the traditional scene, we utilize it to extract information about the system that can later be utilized in the malware distribution/exploitation state. We employ a simple python reverse_tcp payload generated using Msfvenom and execute all post-exploitation information gathering scripts native to Metasploit.

The last category is **Botnets**. This type of malware belongs to the most relevant attacks targeting and consequently utilizing IoT devices with its relevance only rising with the projected significant increase of connected IoT devices to 75 billion until 2025 [SRD20]. Coupled with their high vulnerability, these devices are a prime target for any perpetrator attempting to establish a botnet for large-scale attacks (primarily DDoS) [Kol+17]. As such, the wide variety of different toolsets for this purpose comes as no surprise. We consider three of these tools: Bashlite, Ares, and Mirai. Mirai is the most prominent of these tools. First seen in 2016, some of the most powerful DDoS attacks in history can still be attributed to this malware [Kol+17]. In an attempt to avoid persecution, the original author released the full source code of the bot in late 2016 [Kre17]. This quickly led to not just a high usage of the bot itself but also

the creation of specialized/improved iterations, with many of the IoT botnets being able to trace their roots back to Mirai [Kol+17]. With Mirai, we collect both samples from its dormant state as well as samples of the infected devices performing a synflood against a target in our test setup. We refrain from running similar experiments employing the other botnets as the differences between a synflood performed by either one of the bots should be minimal. Based on functionality and behavior, Bashlite can be considered the predecessor to Mirai. It used to be very popular before the emergence of Mirai and is still relevant today [SRD20]. This can, in large parts, be attributed to the leakage of its source, which — similar to Mirai — led to many spinoffs based on its codebase [Ang17]. Lastly, Ares is a simple python based remote access toolkit and is chosen based on its prevalence in reference datasets [SHG18].

5.4 Pipeline

Software	Version
Pipeline	git-c28dfoaa
Python	3.9.9
NumPy	1.20.3
SciPy	1.7.3
Pandas	1.3.3
Matplotlib	3.4.3
Seaborn	0.11.2
SciKit-Learn	1.0.2
Tensorflow	2.7.0
Pytorch	1.10.1
ThunderSVM	0.3.4
Gudhi	3.4.1
Giotto-TDA	0.5.1
Ripser++	1.1.3
NLTK	3.6.7
TKinter	8.6
Pybind11	2.9.0

Software	Version
Pipeline C/C++ Library	git-d86da953
IPv8 C/C++ Library	git-7e5d7665
Clang	13.0.0
Boost	1.78.0
Eigen	3.4.0
Pybind11	2.9.0

Figure 18: *TDA-Pipeline library versions employed (left: Python libraries; right: libraries used to compile the dynamic libraries of the pipeline).*

The evaluation is performed utilizing Kubernetes orchestrated Docker images running Ubuntu version 20.04 All experiments are conducted using the **TDA-Pipeline**.

The TDA-Pipeline has been in development since 2018 and offers the user a wide variety of processing and classifying methods for an extensive range of different data types. It is based initially on the variant presented in "Machine Learning Applied to Cyber Operations" [BW14] but has since been significantly extended and adjusted to the needs of the graduate and undergraduate students of department 4 of the University of Bonn's institute of computer science. For the purpose of this thesis, the topological data analysis capabilities of the pipeline were extended by two distinct classification methods based on the concept of persistent homology as well as additional TDA feature extraction methods. Figure 18 shows the library versions used during the experiments. Most notable are the topological data analysis modules employed, Giotto-TDA, Gudhi, Ripser, and Ripser++. These were chosen as they outperform competing modules at a significant margin. Ripser/Ripser++ handles the calculation of the persistence diagrams, while Gudhi provides a Python wrapper for Hera, a C++ library that is, to our knowledge — and testing — the fastest for calculating inter persistence diagram distances. Giotto-TDA is utilized to calculate the persistence entropy (cf. section 4.2.1). The pipeline is designed to be deployed alongside a managing server distributing jobs and required data. This server, alongside multiple worker container, are orchestrated using Kubernetes.

5.5 Datasets

We utilize the same sensor setup as well as the same attacks for both datasets (cf. chapter 2 and section 5.3 for details about the sensor setup and attack selection respectively). Specifically, we utilize a 200ms sensor interval paired with an aggregation interval of 10s. Table 7 shows the instance counts for each class present in both datasets. The number of measurements per attack varies compared to other attacks in the dataset. This has two major reasons: Foremost is the variation of attack time as they were manually executed. Additionally, we can observe dropped packages for certain attacks that exhaust the device's resources. This is, however, also behavior that we have to expect in real-world scenarios; therefore, we do not replace or otherwise remediate the lost packages (cf. section 6.4). In the following, we detail the composition of the datasets resulting from both the centralized and distributed usage scenario. To this end, we explore the layout of the data in feature space.

5.5.1 Dataset Centralized Scenario

We are concerned with intrusion detection for homogeneous devices relying on detecting anomalous behavior. Accordingly, the dataset has to capture the behavior of all devices in

Table 7: *Composition of the datasets.*

Attack	Instance count centralized	Instance count distributed
Benign		
benign	896	916
Botnet		
Ares	88	89
Bashlite	85	89
Mirai	86	87
Crypto Malware		
Cryptominer	73	86
Ransomware	75	84
(D)DoS		
GoldenEye	63	66
Hulk	89	84
Slowloris	79	84
Infiltration & Reconnaissance		
Cloud Download	87	86
FTP Patator	79	84
Metasploit	71	83
Nmap	79	87
SSH Patator	83	85

relation to each other. Specifically, we envision intrusion detection to be performed for each device individually by comparing its behavior over the last n time steps (where n can also be 1) to the behavior of all (or a subset) of the other devices in the network. The dataset is designed to capture this data over a period of time from the point of view of a single device. Thus, the respective labels correspond to the label of that specific device. To capture the behavior in relation to the other homogeneous devices, we consider the distance of the respective feature vectors at every aggregation interval. This, however, poses a challenge, as the data types of the sensor data itself are heterogeneous.

In the creation of the datasets, we utilize sensor data aggregation as detailed in section 2.1. The standard aggregators, however, handle only part of the sensors. Namely, the process information, accessed files, and network data are more complex and carry more nuanced information that can be easily lost if not dealt with properly. On top of that, all three of these sensors feature variable numbers of data points at each sensor interval in addition to the multiple — but fixed number — of values per aggregation interval. Thus, every single feature vector consists of floating-point numbers (e.g., CPU load), sets of strings (e.g., accessed

files), lists of dictionaries (e.g., network connections), and dictionaries of dictionaries (e.g., process information per process ID). To process this information, we employ multiple distance metrics as well as topological data analyses, specifically persistent homology. Regarding the floating-point numbers in the feature vectors, we calculate the absolute difference; however, we need more advanced methods for the more complex entries. The Jaccard distance allows us to measure the difference between two sets and thus can be used on the *accessed_files* sensor values. Given two sets A and B it is defined as [SF09]:

$$J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{5.1}$$

This leaves *new_process_info* and *network_activity* each featuring their own unique challenges, which we address with the same fundamental approach, namely persistent homology. We already introduced this methodology in section 4.2.3. Specifically, we can view each entry in both sensors' data as a single unique measurement to create a point cloud describing the device's behavior (from this sensor's point of view), spanning the entire aggregation interval. In the case of the process information, every point describes a single process, while for the network activity, each connection is represented by a point. We capture the device's behavior (i.e., the structure of the point cloud) by computing the persistence diagram of the point cloud. Afterward, we employ the Wasserstein metric to compute the distance between two devices' diagrams (cf. section 4.2.3), giving us a measurement of the difference between the raw sensor values. We encode values represented as strings (e.g., the destination IP address) by assigning them a unique ID for each aggregation interval. In a real-world scenario, this is not advised as it results in a significant loss of information. However, due to the limitations of our experiment setup, these values carry too much information to accurately evaluate the performance of the employed classification methods (e.g., a model can simply learn the attacker's IP). All in all, this results in a dataset with 2022 instances described by 56 features utilized in the evaluation with different sized time intervals, employing a sliding window approach.

Structure

Figure 19 displays the t-SNE embedding of the standardized dataset. There is a distinct cluster formation related to the different classes, validating our sensor selection as well as aggregation and data fusion approaches. Nevertheless, we see some areas where we expect problems during the classification. Primarily these concern the idle behavior of the three botnets (Ares, Bashlite, and Mirai) here shown in blue, light blue, and lavender, as well as cloud download (shown in light orange). All of them feature a broader spread of their respective instances.

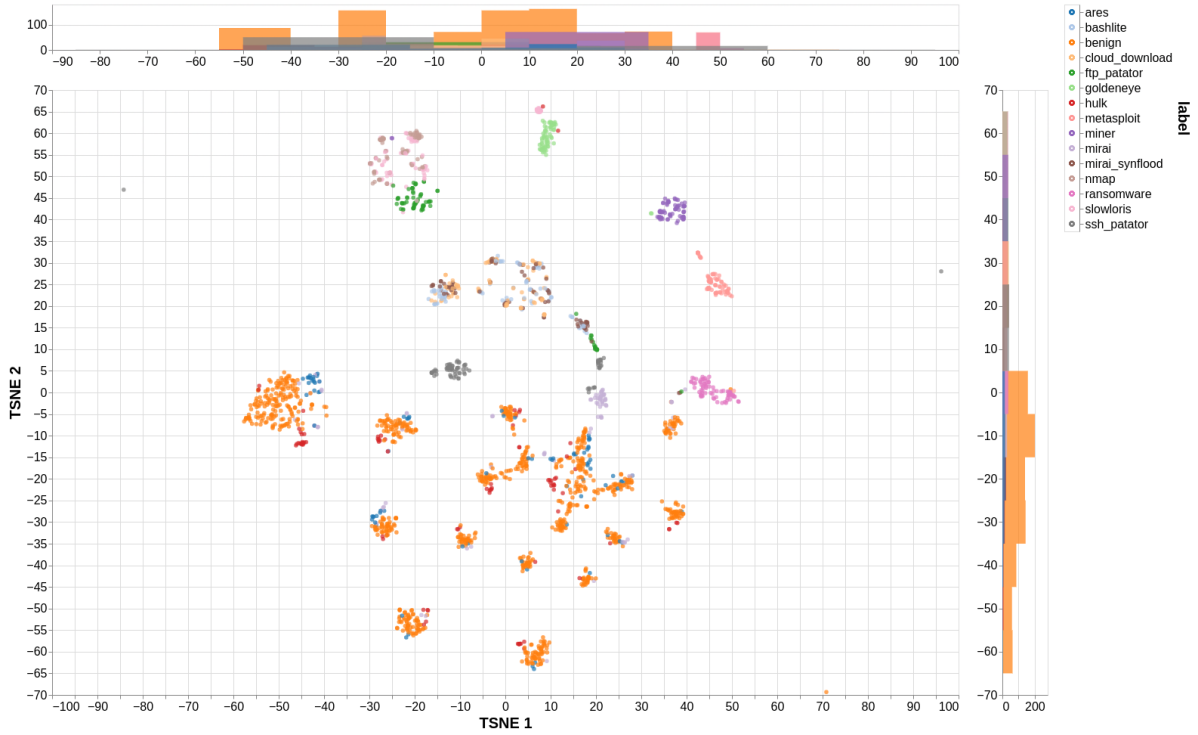


Figure 19: *Two-dimensional t-SNE embedding of the centralized dataset.*

Especially cloud download and Bashlite appear to share many characteristics, with their clusters fully overlapping. Thus, we predict difficulties in establishing decision boundaries that can accurately capture the minute details between these attacks.

A different approach to visualizing high-dimensional data is employing the TDA mapper algorithm. TDA Mapper allows the exploration of the structure of a given dataset. It is introduced by Singh, Memoli, and Carlsson in "Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition" [SMC07]. On a high level, Mapper separates the space into overlapping hypercubes [SMC07]. All data instances are then clustered on a per-cube basis [SMC07]. Afterward, a network is constructed where each node corresponds to one of the clusters [SMC07]. Two nodes are connected by an edge if the respective clusters share members [SMC07]. Projection/Scaling allows for preprocessing before the clustering process.

Figure 20 shows the dataset visualized by the mapper algorithm. Each node represents a cluster (i.e., it can contain instances from multiple different classes), with its color being determined by the label the majority of its members hold. The size of each node is proportional

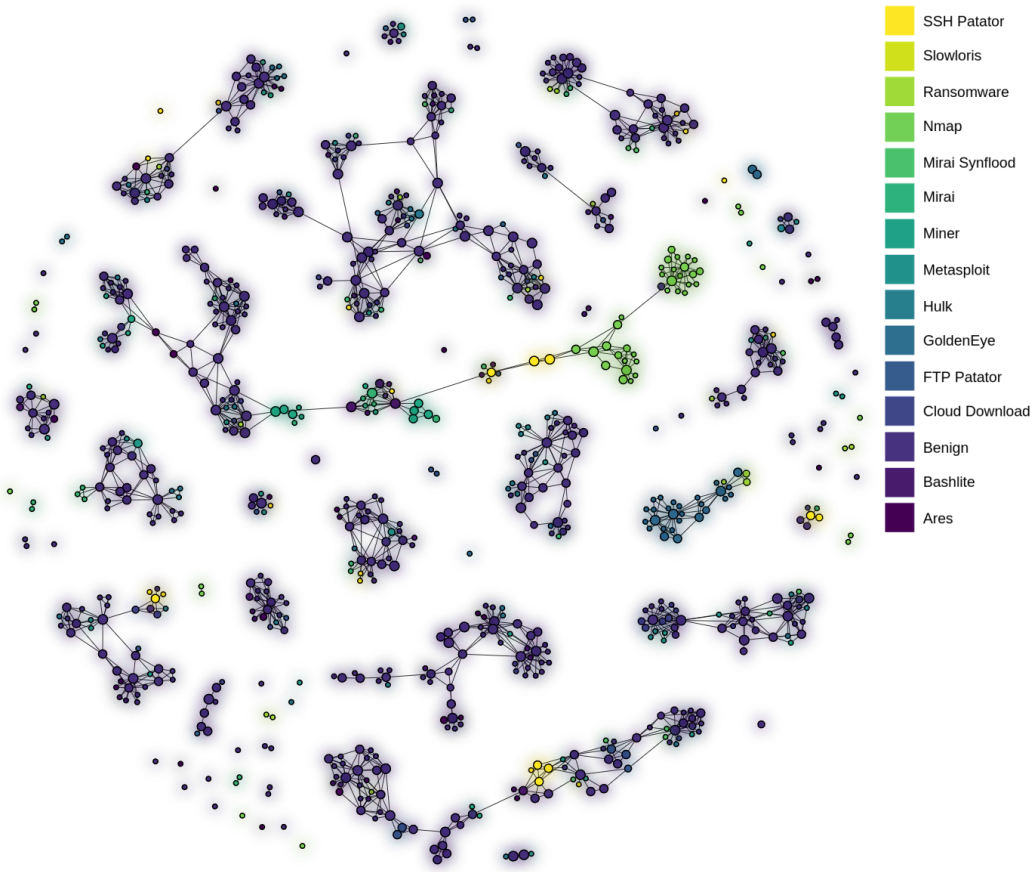


Figure 20: *TDA-Mapper visualization of the centralized dataset (16 hypercubes with 40% overlap, spectral clustering (8 cluster); preprocessed using standardization and feature extraction by t-SNE).*

to the size of the cluster. We can observe several clearly defined cluster constellations that we expect to be correctly classified with high confidence (e.g., Nmap, SSH Patator, Miner, GoldenEye). Nonetheless, we can also spot a high distribution of normal instances, often with single cluster nodes featuring instances from multiple different attack classes. In particular, we observe significant overlap with the classes we already established as difficult to classify in the previous paragraph. Especially for the botnets, we consequently expect not just misclassification with each other but also as benign, which given the context of intrusion detection, is considerably more severe.

5.5.2 Dataset Distributed Scenario

The general setup of the intrusion detection process proceeds parallel to the procedure introduced in section 5.5.1. The key difference is the usage of a distributed detection process. Accordingly, every device receives all other devices' sensor data to allow for detection to take place (cf. chapter 5.2). While this influences the dataset structure, the process of sensor data fusion remains identical to the one introduced earlier. The resulting dataset comprises 2097 instances with 56 features as shown in Table 7.

Structure

Figure 21: *Two-dimensional t-SNE embedding of the distributed dataset.*

Figure 21 visualizes the structure of the dataset following the same procedure as employed for the first dataset. In particular, we standardize the data and afterward embed it in two dimensions utilizing t-SNE. Due to the nature of t-SNE, this allows us to showcase the cluster structure present in the dataset. In direct comparison to Figure 19 we can observe a stark difference between the two dataset's structures. While there are clear, distinct clusters for almost every class in the first dataset, we can observe a significant amount of overlap. This can be attributed to the intrusion detection process. Due to the sharing and processing of the

other devices' data, a significant amount of noise is added to the sensor values. Especially for sensors observing network and disk utilization.

The different impact of the noise on the varying sensors is nicely visible in Figure 21. All attacks with a high impact on sensors other than network or disk activity still showcase clear clustering. Namely, these are attacks with high computational costs for the victim like "miner" or "ransomware", but DDoS attacks against the victim also fall under this category.

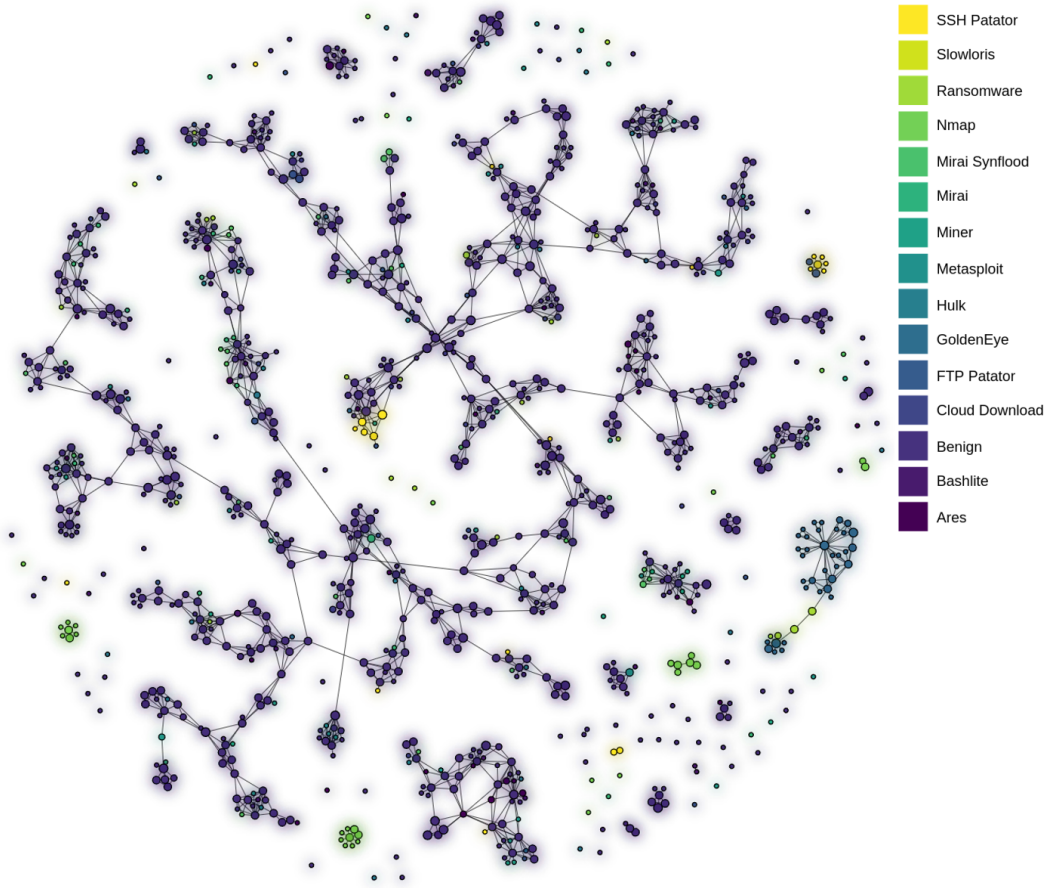


Figure 22: *TDA-Mapper visualization of the distributed dataset (16 hypercubes with 40% overlap, spectral clustering (8 cluster); preprocessed using standardization and feature extraction by t-SNE).*

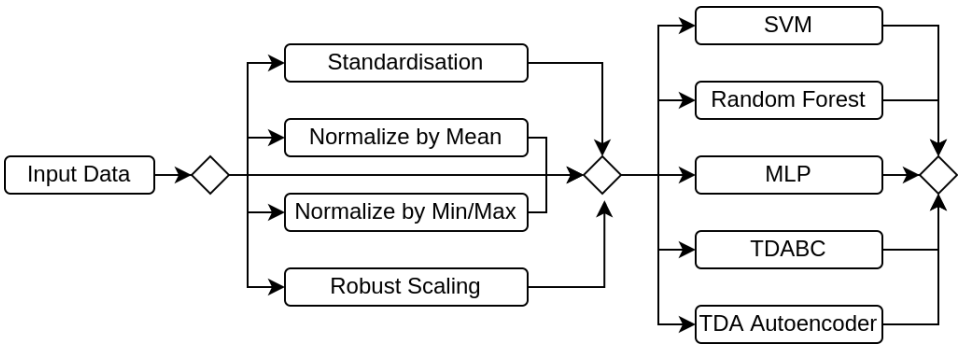
The mapper visualization (cf. Figure 22) shows an interesting picture. Mapper enables the exploration of the topology of a given dataset. This allows us to anticipate how our TDA-based approaches might react to the given dataset. In particular, we can observe that the visualization shares many similarities with the one of the first dataset. Especially concerning the impacted

classes, we can observe the same trends. This is in stark contrast to what we can observe in the t-SNE embedding. In particular, mapper is able to separate out classes that should be heavily impacted by the IDS induced noise (e.g., *Mirai* or *Cloud Download*). This hints at key structural properties that can be picked up by the behavior analysis and mitigate the impact of the IDS traffic and processing.

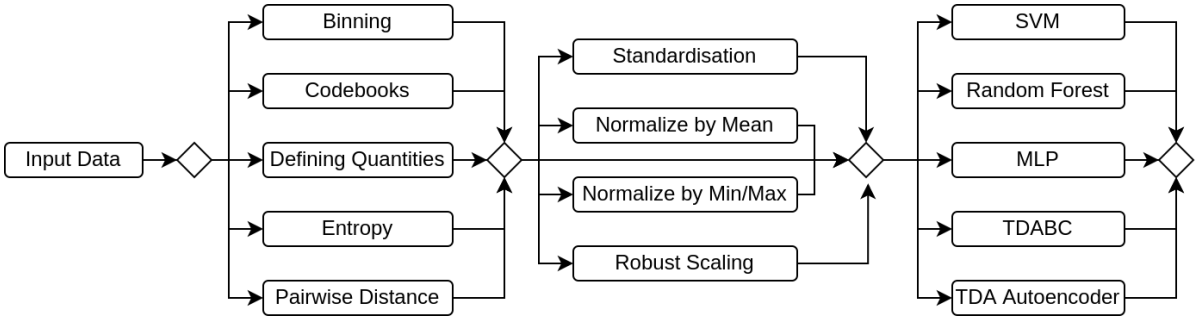
Overall, given the structural properties of the dataset, we expect significantly worse classification performance then what is achievable on the first dataset. Where we just expected a high number of false negatives for the botnet attacks in the first set, here we expect high false positive and negative rates for all non-computational expensive attacks. Thus, we assume simple state-based detection to be insufficient for this scenario. However, behavior-based detection should be able to differentiate between the majority of attacks.

5.6 Classification Procedure

Figure 23 shows the classification procedure chosen. Our primary goal is to understand and illustrate the potential of topological data analysis at various stages of intrusion detection for homogeneous devices. To this end, it is imperative to evaluate the methods on a general but diverse set of use cases. We achieve this mainly by choice of the scenarios; yet, a wide but representative range of classification procedures is also important. Accordingly, we choose our classification pipeline in a way that showcases a selection of simple yet typical procedures utilizing a wide spectrum of distinct methods. The procedure is hereby subject to the stage at which we introduce the TDA methods. With respect to the procedure, this manifests in the nature of the input data. Here we differentiate between a single time step setup (i.e., we only compare device states) and a multi time step one (i.e., we compare device behavior over various time periods). Former features a single feature vector per data instance and can thus directly be preprocessed (cf. Figure 23(a)). In the latter, every data instance is made up of n feature vectors, where n is the number of time steps. To embed this multi-vector input data, we utilize topological data analysis based feature extraction (cf. section 4.2) resulting in the procedure given in Figure 23(b). The topological data analysis based methods (both for classification and feature extraction) as well as the motivation behind choosing them are already discussed in section 4.2 and 4.3 respectively. In the following, we briefly motivate the choice of the remaining preprocessing and classification methods.



(a) Single time step procedure.



(b) Multi time step procedure.

Figure 23: *Graphs visualizing the experimental procedure.*

5.6.1 Preprocessing Methods

The preprocessing methods can be roughly separated into two categories:

1. Anomaly preserving methods
2. Anomaly neglecting methods

For each of these broader categories, we evaluate two methods. Namely, these are robust scaling and normalizing by mean for category one as well as standardization and normalizing by min/max for category two. These specific methods are well suited for our needs, as they are commonly used, with one respectively satisfying the corresponding category criterion strongly, while the other satisfies it weakly. Additionally, we also evaluate the methods without applying any preprocessing.

The selected anomaly preserving methods are robust scaling (strongly preserves anomalies) and scaling by mean (weakly preserves anomalies). On the other hand, standardization (weakly

neglects anomalies) and min-max scaling (strongly neglects anomalies) represent the category of anomaly neglecting methods.

- Scaling by Mean [Boe18]:

$$x_{new} = \frac{x_i}{\max(x) - \min(x)} \tag{5.2}$$

  In this procedure, the 5% values deviating most from the average are disregarded in the calculation of the minimum or maximum value, respectively.

- Robust Scaling [Ped+11]:

$$x_{new} = \frac{x_i - median(x)}{IQR} \tag{5.3}$$

  IQR denotes the interquartile range between the first and third quartile.

- Standardization [MU13]:

$$X_{new} = \frac{X - \mu}{\sigma} \tag{5.4}$$

  Here μ represents the mean and σ the standard deviation.

- Min-Max Scaling [MY10]:

$$x_{new} = \frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{5.5}$$

5.6.2 Classification

Outside of the topology classifiers, we evaluate a total of three classification methods. This set aims not to achieve the best possible classification performance but rather to aid in understanding the benefit that our approach may (or not) bring to the field of intrusion detection. Our goal is not the evaluation of classification methods. Consequently, the methods selected are general and straightforward, in contrast to more specific algorithms that may perform slightly better. The three selected algorithms are SVM (featuring an RBF kernel), Random Forest (RF), as well as a simple multilayer perceptron (MLP) [WCC04; Ho95; Dia+17]. Random Forest, in particular, allows us to assess feature importance for class recognition, which aids in quantifying the significance of the more complex features (e.g., process information) in the single time step use case. The MLP features a single hidden layer, with sigmoid as the activation function. Table 8 details the employed methods as well as the evaluated hyperparameter.

Table 8: *Here shown are all machine learning methods employed, including a short description denoting significant changes/adaptions of the default approach if present. The hyperparameter column lists all evaluated parameter values for the respective classifiers.*

Method	Description	Hyperparameter
SVM	Support vector machine employing an RBF kernel [WCC04].	Evaluated parameters: ▪ *Gamma:* 5 values ($normal(\mu = 0.2, \sigma = 0.075)$) ▪ *C:* 5 values ($uniform(0.5, 2.0)$)
MLP	Simple fully connected neural network with one hidden layer (activation function: sigmoid) [Dia+17].	Evaluated parameters: ▪ *Learning Rate:* 8 values ($uniform(0.01, 0.2)$) ▪ *Number of hidden units:* $(8, 16, 32, 64, 128)$
RF	—	Evaluated parameters: ▪ *Number of trees:* $(4, 8, 16, 32, 64, 128, 256, 512)$
TDAENC	Detailed in section 4.3.2.	Evaluated parameters: ▪ *Learning Rate:* $(0.05, 0.1)$ ▪ *Batch Size:* $(32, 54, 128)$ ▪ *Number of units (first layer):* $(128, 256)$ ▪ *Number of units (second layer):* $(64, 128)$ ▪ *Number of units (third layer):* $32, 64)$ ▪ *Number of units (fourth layer):* $(16, 32)$ ▪ *Number of units (fifth layer):* $(2, 4, 8)$ ▪ *Expected anomalies percentage:* $(55\%, 65\%)$
TDABC	Detailed in section 4.3.1.	Evaluated parameters: ▪ *Interval Selection* $(Maximum, Mean, Random)$

6 Evaluation

The primary goal of the evaluation is to ascertain the usability of topological data analysis for the purpose of anomaly bases intrusion detection, specifically in the field of homogeneous IoT devices. To this end, we separate the introduction of TDA-based methods into three stages. The **first stage** is the introduction of persistent homology based distance metrics to calculate the difference between complex sensor values within the set of homogeneous devices. For this purpose, we consider the state of all devices — as given by the sensor values — at a certain point in time in direct comparison to each other. The goal is to discover anomalous devices states to detect compromised devices. As a **second stage**, we capture the devices' behavior of various time periods by increasing the number of considered time steps. Here we employ TDA-based feature extraction to condense the sensor vectors describing each time interval into a single feature vector suitable for our classifiers. Consequently, instead of just being capable of detecting anomalous devices states, this approach allows us to compare device behavior over a fixed period. The **third stage** evaluates the inclusion of topological data analysis directly into the classification procedure. Here we ascertain the performance of two persistent homology based classifiers for our specific use case, both concerning anomalous device state and anomalous device behavior detection. Next to the usability of TDA-based methods, we also assess its comparative performance, as well as the quality and merit of the sensors introduced by us via comparison to previous work, also concerned with intrusion detection for homogeneous devices [Ten21]. The evaluation steps mentioned above are performed for both scenarios.

We evaluate our results employing the following metrics:

- Precision: $\frac{TP}{TP+FP}$
- Recall: $\frac{TP}{TP+FN}$
- Matthews Correlation Coefficient (MCC): $\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$

TP, TN, FP, and FN correspond to True Positives, True Negatives, False Positives, and False Negatives. We assume the reader is familiar with both precision and recall (in short, recall

measures how good the algorithm is at finding all malicious instances while precision measures its ability to label malicious instances correctly as such). Our primary measurement is the MCC. It measures the correlation between the prediction and the actual label and falls into the interval $[-1,1]$ where -1 corresponds to perfect negative correlation, 0 to no correlation, and 1 to perfect positive correlation. An important characteristic and the main reason for its usage is its invariance to class imbalance (which is very prevalent in two of the three tested datasets) [Luq+19]. All statistics are calculated employing five-fold cross-validation.

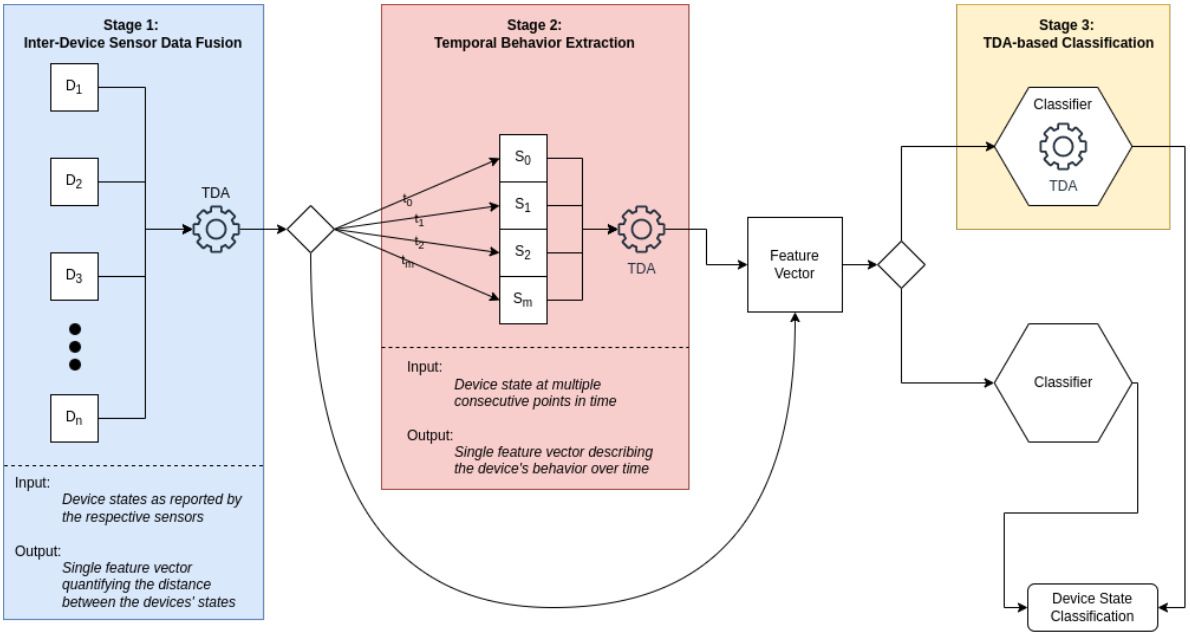


Figure 24: *Application areas of topological data analysis to the detection process.*

The following evaluation is separated into two main parts on the basis of the two scenarios (cf. section 5). Both parts adhere to the same structure, primarily dictated by the aforementioned stages in which we introduce TDA-based methods into the classification process. These stages are visualized in Figure 24. As mentioned earlier **stage 1** facilitates inter-device comparison by providing a distance metric for sensors providing more complex values. Specifically, it only applies to the sensor *new_process_info* and *network_activity*, both providing measurements, that can not be compared using simple methods. **Stage 2** receives multiple device states as input and thus incorporates **stage 1** as we still need to quantify the difference between the individual devices. Lastly, the TDA-enabled classifiers in **stage 3** require a feature vector as input which is provided by the output of **stage 1** or **stage 2**. Accordingly, the possible routes we evaluate are:

1. **Stage 1** → classification
2. **Stage 1** → **stage 2** → classification
3. **Stage 1** → classification by **stage 3**
4. **Stage 1** → **stage 2** → classification by **stage 3**

After formulation of our hypotheses (cf section 6.1), we first assess purely device state based classification (route one; cf. sections 6.2.1 and 6.3.1). Following, we evaluate sensor importance (route 1; cf. sections 6.2.2 and 6.3.2) and device behavior-based detection (route 2), whereby we consider a set of different length time periods and the effect of their length on classification performance (cf. sections 6.2.3 and 6.3.3). As a fourth step, we evaluate the performance of TDA-based classifiers and how they stack up to classic methods both in the device state and device behavior classification (route 3 and 4; cf. section 6.2.4 and 6.3.4). Lastly is the evaluation of TDA-based feature extraction in direct comparison to the T-ChangeDrift system (cf. sections 6.2.5 and 6.3.5).

6.1 Hypothesis

As already discussed in section 5.5, founded in the nature of the attacks, there are pronounced differences in expected classification difficulty reflected in the dataset's structure. Based on this behavior and the behavior of the attacks, we can establish a number of hypotheses with respect to the effectiveness of topological data analysis at the different stages discussed earlier.

> **Hypothesis 1:**
> Inter-device sensor data fusion utilizing persistent homology leads to meaningful and high information feature vectors.

By nature of the use case, our goal is that of anomaly detection. In particular, we search for anomalous behavior whenever a time period is involved. This also holds true on the micro-level, particularly for sensors capturing a snapshot of system behavior every aggregation interval instead of condensing the values into a single system state (*new_process_info* vs. *cpu_load*). Accordingly, we argue that this anomaly should be reflected in the topology of the respective point clouds and consequently be capturable by employing persistent homology. We assess this hypothesis by evaluating classification performance and feature importance in sections 6.2.1, 6.3.1, 6.2.2, and 6.3.2.

> **Hypothesis 2:**
> Our sensor selection, specifically the additions in contrast to T-ChangeDrift facilitated through TDA-based inter-device comparison allows for good classification, especially with respect to difficult classes such as dormant botnets.

We argue that complex threats not showcasing blatantly obvious anomalous behavior (e.g., dormant botnets) require more advanced sensors to capture the fine-grained differences to normal user behavior. Consequently, simple device hardware state sensors are insufficient and must be complemented by sensors observing the operating system's behavior and active services. We aim to prove this hypothesis by evaluating our classification performance with respect to the sensors in question in sections 6.2.2, and 6.3.2, while discussing feature importance in relation to both device state and device behavior classification in section 6.2.3 and 6.3.3.

> **Hypothesis 3a:**
> Topological feature extraction on time-series data retains necessary information to facilitate classification. The resulting behavior vectors show a clear information gain compared to simple state vectors with improved classification performance featuring clear dependence on the number of time steps

A significant portion of IoT malware are botnets that spend substantial periods dormant. As such, we expect behavior analysis to provide benefits over state analysis by evaluating sensor measurements over a fixed time period. Additionally, minor anomalies and deviations in the behavior of singular devices are to be expected due to external or internal influences. Timeseries classification introduces the necessary robustness to identify and handle these outliers. We discuss this hypothesis in detail in sections 6.2.3 and 6.3.3.

> **Hypothesis 3b:**
> Topological feature extraction on time-series data results in quantifiable higher information retrieval than what T-ChangeDrift is able to achieve.

ChangeDrift is an abstract concept regarding the detection of the anomalous behavior of any sort within a system of homogeneous nodes. As mentioned earlier, we are considering Tennié's ChangeDrift (T-ChangeDrift) an IDS with the goal to detect anomalous behavior for homogeneous IoT devices. To this end, it relies on hierarchical agglomerative clustering to extract behavioral information from the devices. We hypothesize that a TDA-based approach can extract greater amounts of information that lead to meaningful classification improvements. Agglomerative clustering share many similarities with persistent homology regarding zero-dimensional topological features. We argue that additional observing higher-

dimensional features provides a more completely representation of the data's topology and thus provides meaningful informations. We aim to prove this hypothesis by directly comparing the performance of both systems on the same data (cf. sections 6.2.5 and 6.3.5).

6.2 Centralized Intrusion Detection

The first scenario is concerned with a setup featuring a centralized IDS and is detailed and motivated in section 5.1 and section 5.5.1. The utilized sensor and aggregator combination is shown in Figure 16 and Figure 17. Notable is the absence of the system-call sensor. We clarify this decision in section 6.4 The classification procedure follows the process given in diagrams 23. In the following, we evaluate the merit of TDA at various stages of the detection process. To this end, we utilize the classification procedures to gain an understanding of the method's behavior in a variety of different settings. The sections follow the different stages, with each additionally referring to the hypotheses.

6.2.1 Anomalous State Detection

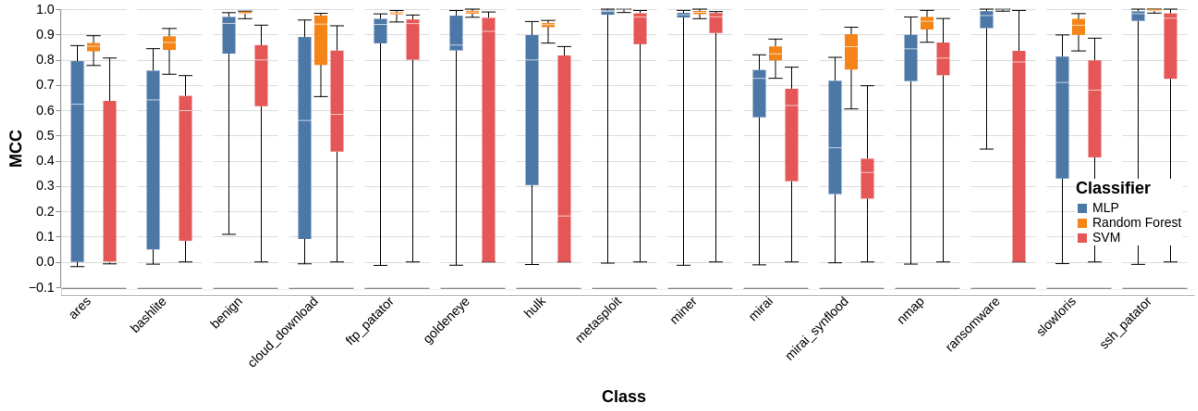


Figure 25: *Boxplots visualizing the per class classification performance per classifier as given by the MCC. Whiskers show min-/max-values.*

Anomalous state detection is concerned with the system state as reported by the sensor data for all homogeneous devices. A classification is achieved by comparison of device states (cf. section 5.5). Thus, this is **stage** 1 of the introduction of TDA into the detection process. As mentioned in section 5.6 we consider a total of three non TDA classifiers with a variety of different preprocessing methods (cf. Figure 23(a)).

Figure 25 visualizes the MCC per class per classifier. Each classifier's performance is visualized as boxplots based on the different results achieved depending on chosen preprocessing and hyperparameters. Here the whiskers show the respective minimum and maximum values, as, by the nature of the data, we do not have values that can be considered abnormal. It is clearly visible that the Random Forest classifier outperforms the rest, both in terms of maximum and minimum MCC achieved. MLP reaches the second-best results, while SVM performs worst. For Random Forest, the best performance was reached when using 128 trees without any prior preprocessing of the data. This leads to an MCC of 0.97. The MLP achieves its best result (MCC of 0.92) with robust scaled data, 64 hidden units, and a learning rate of ~ 0.1367, while SVM requires standardization, a C of ~ 0.2491, and γ equal to ~ 1.6170 to reach an MCC of 0.83. Additionally, the spread between maximum and minimum values is considerably smaller for the Random Forest classifier. This is anticipated as Random Forest is largely invariant to data scaling as it is in contrast to SVM and MLP, not distance-based.

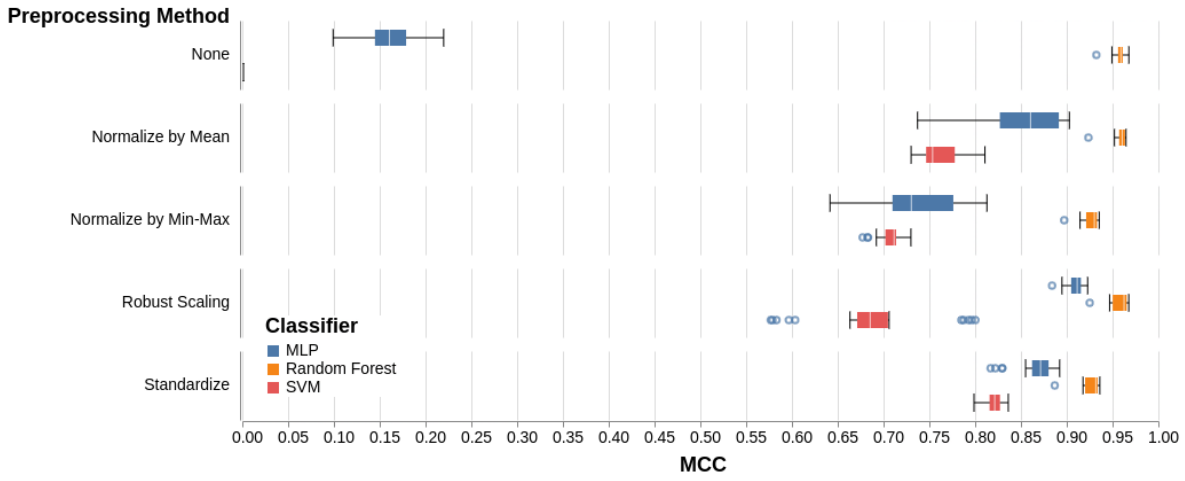


Figure 26: *Boxplots visualizing the impact of preprocessing choice on classification performance. Whiskers show upper and lower quartile.*

A different point visible in Figure 25 is the class-dependent differences that we hypothesized about in section 5.5. In particular, we can observe a clearly worse performance when dealing with the three botnets for all classifiers. This is caused by the significant similarities these attacks share with each other and with the benign instances.

Figure 26 showcases the average MCC reached by the different classifiers in relation to the preprocessing method. It clearly shows that the main reason for the behavior observed in Figure 25 can be attributed to significant differences in value range per feature, which consequently implies a heavy reliance on normalization for any distance-based classification

approach, as evidenced by the poor performance of both MLP and SVM if none such scaling is performed. We can also observe a better performance of the MLP over the SVM across almost all tested methods. The deviation in the performance of the SVM model using robust scaling is noteworthy. The behavior can be attributed to the higher reliance on feature scaling of support vector machines. The figure also showcases the relatively low importance of hyperparameter choice in contrast to preprocessing (visible in the extent of the boxplots). As discussed prior, this is to be expected for Random Forest; however, it is remarkable for the MLP classifier. Nevertheless, we can observe a correlation between hyperparameter caused MCC spread and employed scaling method. This can be attributed to the sensitivity of these algorithms to feature scale.

Interesting is also the difference in classification performance between the preprocessing methods. Here we can observe a clear dip in performance if the data is scaled by min-max. This indicates the presence of meaningful outliers within the feature data. However, given the performance of the SVM in combination with robust scaling, we can assume that the number of outlier values is limited. This assumption is confirmed in our evaluation of the per-class feature importance (cf. section 6.2.2). There we can observe a clear difference in scaling for the features *network_activity* and *new_process_info*, which, coupled with the RBF kernel of the SVM, leads to an imbalance between relative feature weights and their actual importance. This results in the SVM classifier's deviating behavior, as also observed here.

Assessment

The evaluation of the general performance of the classifiers for anomalous state detection validates a number of hypotheses we made prior. In particular, we can observe a clear correlation between attack class and classifier performance, as hypothesized. In combination with the good performance of the classification, this strongly hints that our first hypothesis **H1** should hold. We further evaluate this claim by assessing the sensor importance in the following section. Additionally, we observe deviations in the behavior of the SVM classifier, indicating a suboptimal preprocessing for that specific model. This is also explored in greater detail in section 6.2.2. All in all, the performance of all classifiers is satisfying and validates the merit of our approach.

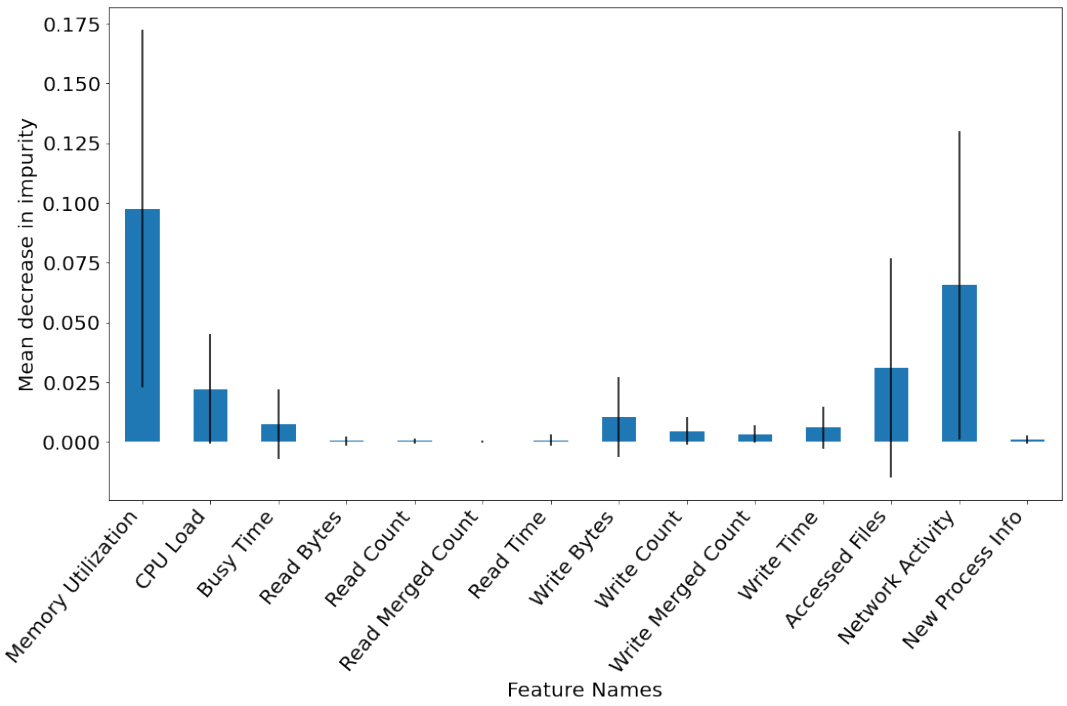


Figure 27: *Barplot showing the mean decrease in impurity per feature as reported by the best performing Random Forest model (errorbar shows standard deviation) [Ped+11].*

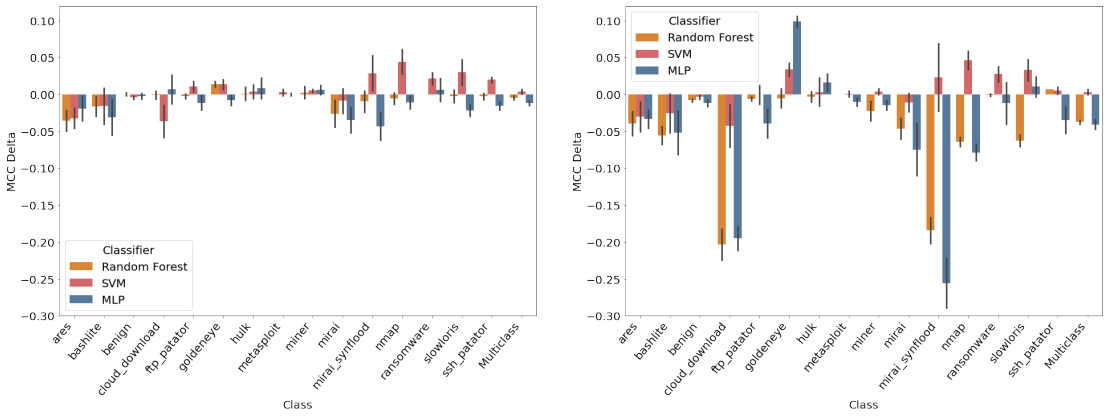
6.2.2 Sensor Importance

As discussed in detail in section 2.4, our work extends and adapts on a previous work called T-ChangeDrift [Ten21]. From this work we adapt the sensors *cpu_load*, *memory_utilization* and *accessed_files*. Consequently, we will not discuss their importance in detail and instead encourage the reader to consult the original work. This leaves the sensors *network_activity*, *new_process_info* and *disk_utilization*. Latter is aggregated by mean and split into individual values (e.g., "Busy Time", "Read Bytes"), while the other two utilize TDA in the sensor fusion process, and thus we expect some degree of information loss. However, we hypothesize that the information loss is insignificant enough such that *network_activity* and *new_process_info* are still of high importance, especially for attacks operating primarily covert. On the other hand, we expect them to only offer little additional information for other attack classes. In the following, we discuss general feature importance followed by a deeper dive on the TDA-enabled features with a special focus on **H1** and accordingly extending our understanding of **stage 1**.

General Sensor Importance

One of the classifiers employed by us is Random Forest. The decision trees that are part of the forest are constructed based on the reduction of impurity a certain feature is capable of providing. As such, we can utilize this information to determine feature importance by the mean decrease of impurity over all trees in the forest [Ped+11]. The results are given in figure 27. Here the bars show the mean decrease in impurity, with the error bars indicating the standard deviation. Clearly visible is that the more hardware-focused sensors already found use in T-ChangeDrift also hold high importance here. This is due to the majority of attacks not attempting to act covertly. Thus, sensor values like the CPU utilization have a high value, as we can expect a significant deviation from normal behavior. Nonetheless, the high standard deviation indicates that feature importance is significantly dependent on the class. Next to the hardware-focused sensors, we can also deduct the importance of the network activity. This is to be expected, as the majority of the attacks feature some form of network traffic. Yet this still indicates a successful information retrieval by our topology-based approach. The impact of new process information appears to be low, but this requires some further investigation. Lastly, *disk_utilization* (split into its sub-values) appears to provide only moderate amounts of information, which can be attributed to its overlap with *accessed_files*.

Sensor Importance and Information retention of TDA-based sensor data fusion



(a) Feature: *new_process_info*

(b) Feature: *network_activity*

Figure 28: *Barplots visualizing the average MCC difference between the dataset with and without specific sensors enabled (MCC with sensor minus MCC without sensor).*

With *new_process_info* and *network_activity* we introduced two sensors that we argue are beneficial for the purpose of detecting silent malware, especially botnets in their dormant

state. These sensors, however, deliver complex sensor data and thus require topological data analyses to perform inter-device comparisons. As such, it is essential to evaluate their impact with regard to the specific classes they are designed to handle. To this end, we have to consider if the sensor itself has merit and if the TDA-based extraction captures all relevant information. To achieve these goals, we reevaluate the performance per classifier while omitting one of the respective sensors (cf. section 6.2.1). We perform this experiment ten times per classifier and then calculate the per class MCC Δ (MCC with sensor - MCC without sensor) with respect to the original results.

As we have established in the previous sections, the majority of the classifiers are impacted by the choice of preprocessing. Since the optimal method is influenced by reevaluating performance with certain sensors deactivated, we choose to measure the performance Δ between the values achieved employing the best performing preprocessing for the reduced dataset and the respective classifier in comparison to the values achieved using this preprocessing on the full dataset.

Figure 28 displays the results as bar plots, with the standard deviation shown as an error bar. The graph clearly shows a correlation between the new features and the classification performance of the classes in question. For all three classifiers, we can observe a clear decrease in the MCC for the classes Ares and Bashlite for both *new_process_info* and *network_activity*. These two attacks only deviate minutely from benign behavior. Accordingly, their detection requires fine-meshed sensor measurements, in particular regarding network traffic. These are given with the two respective sensors; however, this information also has to be extracted by the TDA-based sensor comparison. The clear negative MCC Δ shows that this information was successfully encapsulated in the distance measurement and proves the merit of our approach. This is especially promising with regard to the persistent homology based feature extraction of device behavior that we evaluate in section 6.2.3. Furthermore, we can observe — on a per-class level — the same behavior as in Figure 27 with *network_activity* holding higher importance than *new_process_info*. This is easily explainable by the nature of the attacks. Other than "Miner", "Ransomware", and "Benign" — for neither of which we can observe a difference in importance between the two features — all attacks require in some form network traffic. This results in a high value of the sensor data provided by observing the network activity.

An additional attack that operates largely inconspicuously is cloud-download. Given its nature, we expect the *network_activity* sensor to provide high-value information. This is confirmed in diagram 28(b). Inclusion of this feature results in an up to 0.2 points better MCC regarding this specific class (depending on the chosen classifier), additionally affirmed by the

respective low standard deviations. Interesting is the performance with regard to the third botnet, Mirai. The classification performance of both MLP and Random Forest clearly suffers; however, for the SVM, the result is less clear. Another attack that should benefit from the *network_activity* feature is a synflood carried out employing Mirai. This holds true for MLP and Random Forest, with an MCC gain of up to 0.18 and 0.25. Yet SVM showcases an opposite behavior again, with an increase in performance when not considering network activity.

Intriguingly, we can only observe this behavior of the SVM classifier when utilizing standardization. All other preprocessing methods result in behavior very similar to the one of the MLP and Random Forest classifiers. Therefore, we hypothesize that it is caused by the kernel. We employ an RBF kernel as this is the standard choice of a kernel for non-linear SVMs. The RBF kernel, given hyperparameter γ and feature vectors x and x', is defined as:

$$e^{-\gamma\|x-x'\|^2} \tag{6.1}$$

Accordingly, all features are evenly weighted. This is the primary reason why we can observe the terrible performance of the SVM on unnormalized data (cf. Figure 26). The imbalance in feature scale results in a high relative feature importance for features with larger variance. In our case these are both *new_process_info* and *network_activity* featuring a value range of $7.07106e^9$ and $707107e^9$ respectively. This does not correspond to the actual feature importance as indicated by Figure 27. Standardization alleviates this situation, but does not eliminate it. If we consider the difference between the value range of the most important feature for general classification (according to Figure 27) — *memory_utilization* — and the least important — *new_process_info* —, after standardization, we can still observe a difference by a factor of about 12.

Given all this, what we can observe in Figure 28 is the relative weights' improvement outweighing the loss of information for the SVM classifier. Remarkable is that even with this effect, removing either of those features results in a loss of classification performance, especially for the botnet attacks. This emphasizes the importance of introducing these features for this specific classification task. It might appear unintuitive that — as mentioned earlier — we can not observe this behavior for both normalization by mean and robust scaling. This is primarily due to the distribution of feature values for *new_process_info* and *network_activity*. For the latter, the high distance values are clear outliers. Consequently, their outlier nature is preserved by both preprocessing methods (cf. section 5.6.1). This is not the case for *new_process_info*. Here there is a significantly higher number of extremely high values, and hence, both robust scaling and scaling by mean fail in preserving their anomalous nature. This

consequently leads to a collapse of the value range and, accordingly, low relative weight in the SVM classifier. Additionally, the sensor's importance for general classification — as determined by the Random Forest — is the second-largest for the network activity sensor. Therefore, a high relative weight does not impact the performance as much as for *new_process_info*. Accordingly, we can not observe the same behavior when omitting a feature while comparing the results of these methods.

Hence, we can infer that logarithmic scaling ($x' = \log(x_i + |\min(x)| + 1)$ [Haw+02]) before employing any further processing should be beneficial for the SVM classifier. A reevaluation employing the entire feature set leads to a 0.01Δ (MCC of 0.85). As predicted, the improvements primarily result from better class prediction for classes that highly rely on sensors like memory and CPU utilization (Ransomware: $\Delta 0.13$, SSH Patator: $\Delta 0.04$, FTP Patator: $\Delta 0.03$, GoldenEye: $\Delta 0.02$).

Assessment

The evaluation of the sensor importance shows that **H1** holds while also indicating the same for **H2**. In particular, we can observe not only a high reliance on the newly introduced sensors for primarily covert acting attacks but also a high reliance on the *network_activity* in general. The high importance of these features also clearly shows the successful retention of the majority of relevant information from the raw sensor data employing persistent homology (as predicted in **H1**). Given the excellent results of, in particular, Random Forest, we thus consider the state-based classification employing TDA-based sensor data fusion as successful.

6.2.3 Anomalous Behavior Detection

In contrast to anomalous state detection, anomalous behavior detection focuses on a time interval instead of a specific point in time. Each aggregated sensor value corresponds to a single time step describing the device's state during the aggregation interval. Thus, a point cloud of aggregated sensor measurements defines a device's behavior. Similar to anomalous state detection, we aim to discover deviations of a subset of the devices from normal behavior, where normal behavior is defined as the behavior of the majority. As we are dealing with point clouds, behavior comparison amounts to the comparison of the respective structures or, in other words, their topology. Thus, we employ topological data analysis based feature extraction methods to capture this structure in a single feature vector (cf. section 4.2). This corresponds to **stage 2** of Figure 24 followed by a standard classifier.

In the following, we evaluate two key aspects. First is the influence of the time window's length on per class classification performance, and secondly, the different feature extraction methods' ability to capture the devices' behavior. To this end, we first consider the general performance followed by an in-depth assessment of every individual extraction methods.

General Performance

Table 9: *The table shows the best MCC achieved per class depending on different classifiers.*

Time steps	Ares	Bashlite	Benign	Cloud Download	FTP Patator	Goldeneye	Hulk	Metasploit	Miner	Mirai	Mirai Synflood	Nmap	Ransomware	Slowloris	SSH Patator
MLP															
5	0.04	0.12	0.39	0.05	0.61	0.28	0.10	0.45	0.09	0.20	0.11	0.06	1.00	0.77	0.41
15	0.52	0.63	0.59	0.57	0.78	0.71	0.51	0.73	0.60	0.72	0.60	0.58	1.00	0.60	0.60
25	0.85	0.81	0.78	0.78	0.76	0.83	0.69	0.86	0.82	0.84	0.85	0.84	1.00	0.77	0.64
35	0.90	0.91	0.89	0.89	0.93	0.91	0.91	**0.97**	0.93	0.95	0.93	0.95	**1.00**	0.82	0.82
50	**0.95**	**0.94**	**0.90**	**0.95**	**0.97**	**0.95**	**0.97**	0.96	**0.98**	**0.97**	**0.97**	**1.00**	0.99	**0.97**	**0.91**
Random Forest															
5	0.52	0.41	0.52	0.40	0.76	0.49	0.33	0.57	0.42	0.56	0.40	0.36	0.99	0.79	0.65
15	0.83	0.89	0.76	0.88	0.95	0.88	0.83	0.90	0.85	0.90	0.85	0.84	0.99	0.87	0.90
25	0.98	0.95	0.91	0.96	0.96	0.99	0.97	0.98	0.94	0.98	0.96	0.96	1.00	0.97	0.98
35	0.99	0.99	0.92	0.98	0.99	0.99	0.98	**1.00**	0.99	1.00	0.99	0.99	1.00	0.98	**1.00**
50	**1.00**	**0.99**	**0.96**	**0.99**	**1.00**	**0.99**	**0.99**	0.99	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.99
SVM															
5	0.23	0.38	0.37	0.30	0.69	0.28	0.27	0.45	0.27	0.28	0.36	0.30	1.00	0.75	0.54
15	0.61	0.74	0.54	0.56	0.88	0.68	0.54	0.78	0.67	0.75	0.65	0.73	1.00	0.69	0.75
25	0.90	0.88	0.77	0.89	0.89	0.91	0.79	0.86	0.87	0.92	0.85	0.89	1.00	0.89	0.87
35	0.94	0.91	0.81	0.90	0.93	0.95	0.92	0.97	0.96	0.95	0.90	0.92	**1.00**	0.90	0.91
50	**0.96**	**0.97**	**0.90**	**0.96**	**0.98**	**0.98**	**0.96**	**0.97**	**0.98**	**0.99**	**0.99**	**0.99**	0.99	**0.97**	**0.96**

We evaluate a total of five different time window lengths, specifically 5, 15, 25, 35, and 50 time steps. For each time window, we run through all possible combinations given in the classification procedure (cf. section 5.6). This allows us to evaluate the impact of the time window length, independent of possible preprocessing or hyperparameter dependencies. The highest MCC reached per classifier per class is given in Table 9. Highlighted is the best result per classifier, with the overall best per class being additional underlined.

It is immediately apparent that time window length is highly correlated to the classification performance. For all classifiers, the best per class performance is — with a few exceptions —

reached employing the largest time window with a total of 50 time steps. All exceptions to this perform best with the next smallest window of size 35. Additionally, the performance difference for these outliers to the longer time window of 50 steps is minimal. Consequently, this performance indicates that there might be merit for even larger time windows. However, the calculation of persistent homology scales strongly with the number of instances (n) with time complexity of $O(n^\omega)$, where ω is the matrix multiplication exponent [BP19]. Additionally, with 50 time steps, we already compare device behavior over a period of $\sim$ 400 seconds. Consequently, we deem 50 a reasonable upper bound. Furthermore, larger periods can lead to attack overlap in our specific dataset.

With respect to multiclass performances, all three classifiers achieve their best result a 50 time steps. Once again, the best performer is Random Forest with an MCC of 0.98, followed by SVM (0.94) and MLP (0.92). In all three cases, we observe a significant positive Δ in comparison to the device state classification (especially regarding covert attack classes). This indicates a good representation of device behavior in feature space by the TDA-based extraction methods. We explore this further in section 6.2.3. Additionally, we can observe a correlation between the number of time steps and the best performing extraction methods. All three classifiers achieve better results with the codebook or pairwise distance (in particular with the Wasserstein distance metric) in small-time-windows, while binning and defining quantities outperform all other methods when considering longer time periods. We can also observe a weak dependency on preprocessing methods, with anomaly neglecting methods outperforming their counterpart with increasing time window size. Most likely, this is caused by the shift in best-performing feature extraction methods.

Interesting is also the class dependant performance differences. Here we can once again observe the differences between the primarily covert acting and clear abnormal attack classes. Former appear to profit considerably more from larger time windows than the latter. This matches what we observed during the state detection. Consequently, this indicates good information retention, as we already achieve high classification performance with only a few time steps, meaning a larger amount of information has to be drawn from the individual states. The exception to this behavior is the benign instances, which we further investigate in the following sections.

Figure 29 visualizes the classification performance depending on the different feature extraction methods and classifiers. It once again showcases the significantly better performance of the Random Forest classifier relative to all others. However, it also shows clear performance differences between extraction methods. In particular, the feature extraction employing

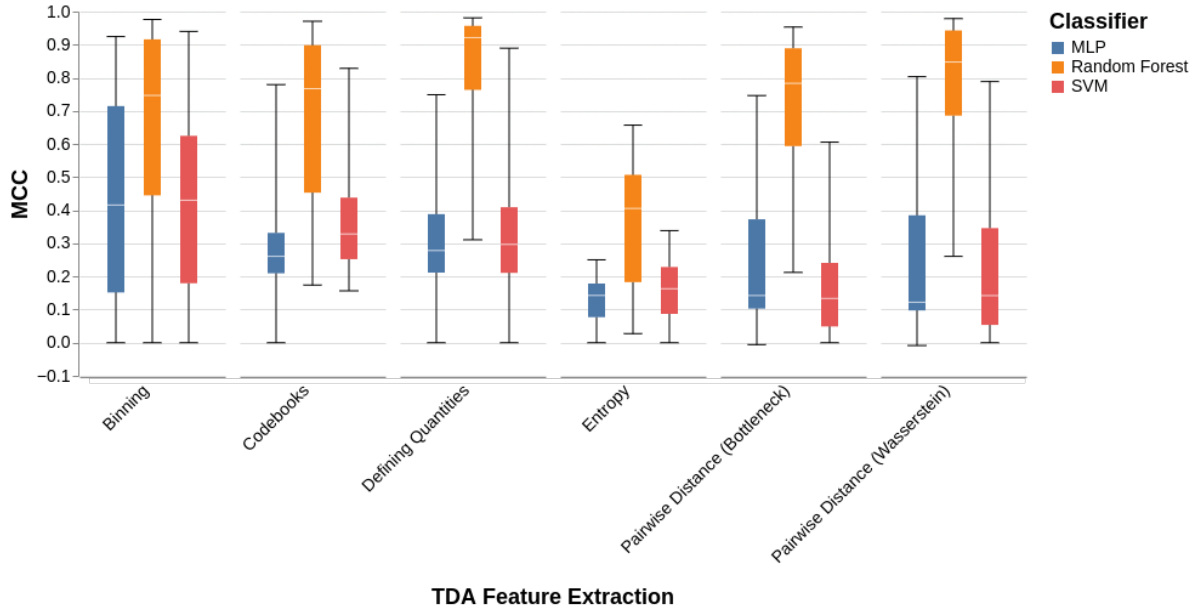


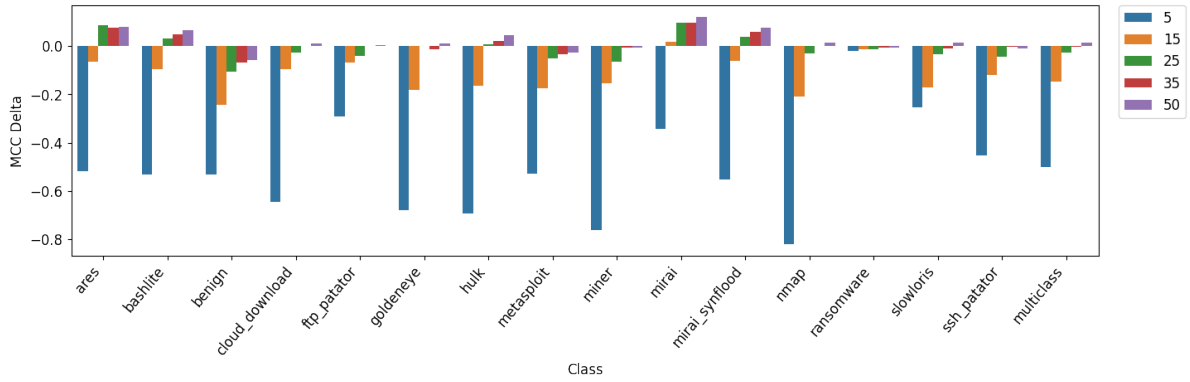
Figure 29: *Boxplots visualizing classification performance (by MCC) depending on feature extraction and classifier choice. Whiskers show respective min-/max-values.*

persistent entropy does not appear to be able to retain the necessary information. In general, the classifier appears to favor more fine-grained methods such as "Binning" that directly capture the topological feature's lifetimes without further processing. This implies the high importance of minute differences between the different classes' behavior.

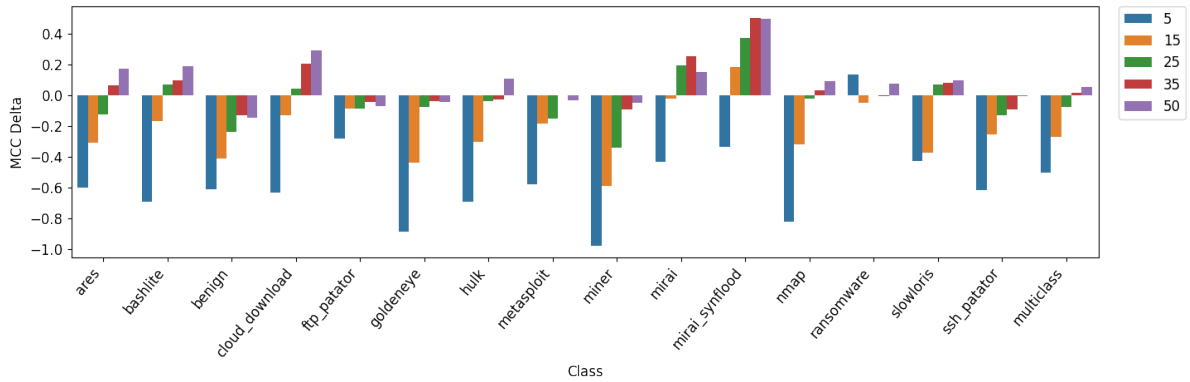
Feature Extraction by Defining Quantities

Regarding feature extraction, we are concerned with two main criteria. First is **information retention**. This refers to the method's capability to capture the device's behavior over time, as reported by the sensor system. The second point is **class separation**, which is — as our sensor provides relevant information (cf. section 6.2.2) — correlated with the first one. The goal is to quantify how well the extraction method facilitates the respective classifiers.

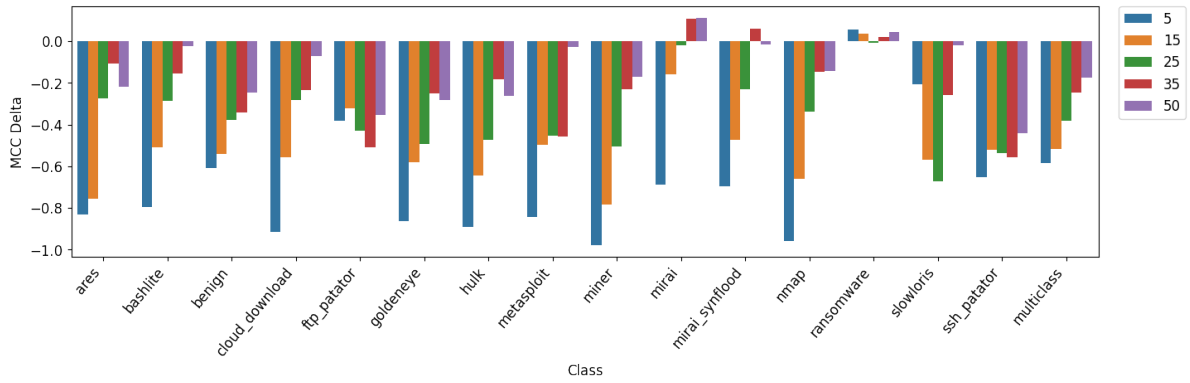
We start by considering **information retention**. This criterion is hard to measure, as we can not compare results by the same classifiers but without feature extraction, as the classifiers do not accept a time series as input. Thus, we put the results in comparison to state detection. As the different feature set impacts optimal preprocessing and hyperparameter choice, we only compare the best state classifier with the respective best behavior classifiers (depending on the time window size), as given by the multiclass MCC. Figure 30 visualizes the performance Δ



(a) Random Forest



(b) SVM



(c) MLP

Figure 30: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction by defining quantities.*

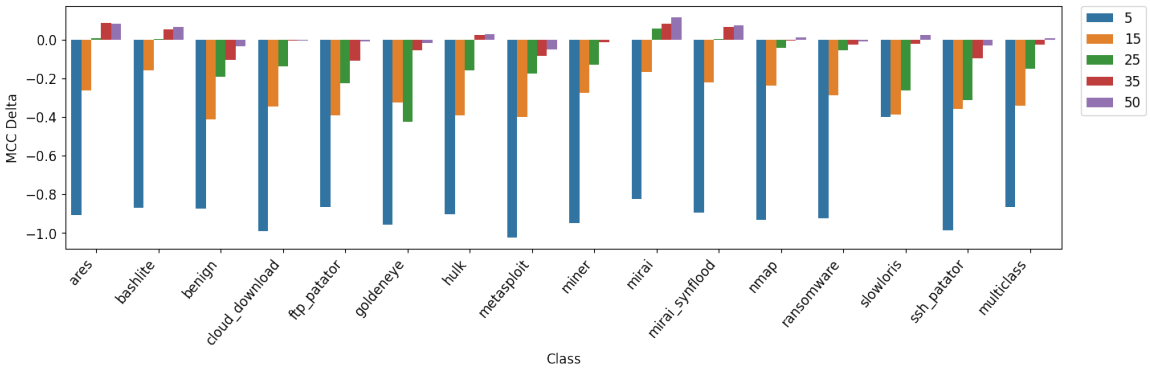
(behavior-analysis MCC minus state-analysis MCC) depending on the number of time steps and classifier chosen. The plot clearly shows the correlation between window size and classification performance.

There are several patterns visible for all three classifiers. In general, window sizes below 35 lead to results worse or equal to the ones achieved utilizing only the device state. This indicates a significant information loss that can only be compensated by observing the device's behavior over extended periods of time. An exception to this is covert attacks. As stated previously, these are significantly harder to detect. Consequently, they require multiple factors to detect while, e.g., a ransomware attack can be sufficiently detected employing a single time step observing only accessed files, memory, and CPU utilization. This behavior is likely to change with smaller aggregation intervals, as they directly impact the robustness of measurements like the CPU load. With smaller aggregation intervals, spikes in, e.g., the CPU load during normal behavior would carry over in the sensor measurement. Accordingly, the sensor data contains more noise, impacting the quality of information the sensors can provide in a single time step. However, the robustness of these values is desired and leads to meaningful detection improvements for the state classification. Therefore, given the chosen aggregation interval, the additional information provided by the behavior analysis easily outweighs the sensor information loss for covert attacks. For the other classes, the sensor information loss proves to be quite detrimental and can only be recovered by large time windows.

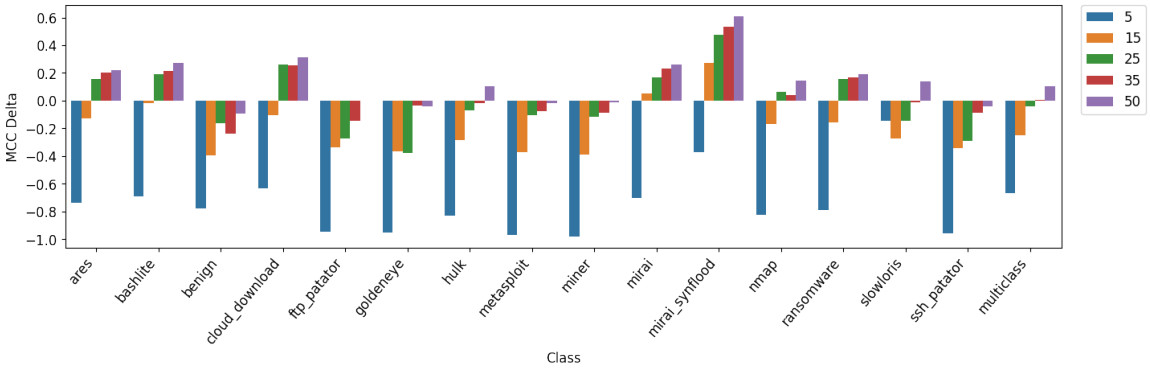
A further notable observation is the clearly worse performance of the MLP classifier. Accordingly, there is a correlation between chosen feature extraction and classifier choice. This is to be expected, as we evaluate a set of very different classifiers that consequently have different requirements regarding the feature space.

The most straightforward way to assess **class separation** is classifier performance. As we already discussed information retention, we focus on the performance achieved employing 50 time steps. As visible in Figure 30 for both SVM and Random Forest, we achieve better results than the already excellent ones achieved during state detection. In particular, Random Forest indicates good class separation, as its decision boundaries are by design parallel to the axis and thus considerably more restrained than, for example, the MLP. While we can observe worse performance for the MLP, this can be attributed to unsuitable embedding rather than insufficient class separation. A metric capable of quantifying class separability is the separability index (SI) [MM08]. It measures the fraction of instances that have the nearest neighbor with the same label [MM08]. Since preprocessing by nature is a transformation of the feature space, we evaluate this metric with respect to only the extraction method. This

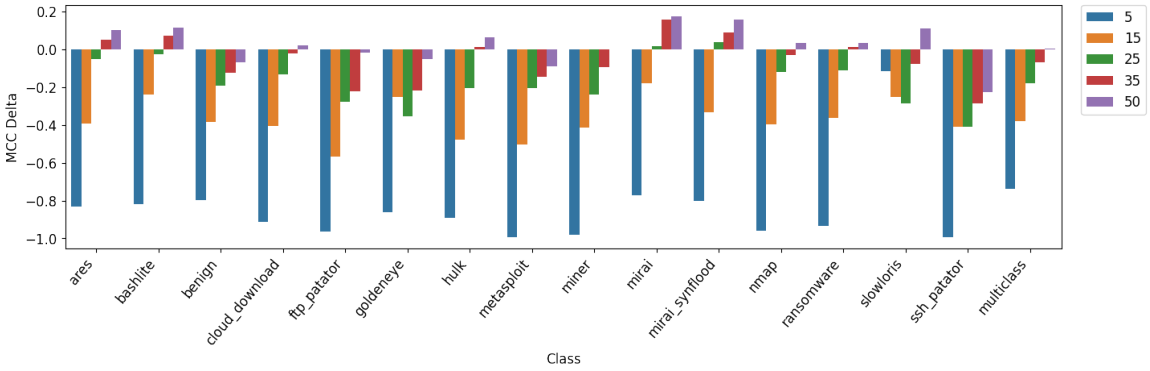
comes out to a value of 0.94 for defining quantities. In comparison, for the unprocessed state dataset, the SI is 0.37. Consequently, it is capable of providing an excellent class separation.



(a) Random Forest



(b) SVM



(c) MLP

Figure 31: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction by binning.*

Feature Extraction by Binning

Figure 31 visualizes the MCC Δ (behavior-analysis MCC minus state-analysis MCC) for the feature extraction by binning depending on the respective classifiers. Regarding **information retention** we can observe almost identical behavior to the defining quantities based extraction. Specifically, there is a clear correlation between window size and performance as well as a significant improvement for covert attacks, both caused by the same issues noted earlier. A difference compared to defining quantities is the improved performance of the MLP classifier, which indicates a more suitable embedding.

General performance is — as with defining quantities — excellent, indicating good **class separation** (cf. Figure 29). In particular, the separation index is with 0.98 notable better than its defining quantities counterpart. This showcases the excellent separability power of the extraction method, which is also hinted at by the good classifier performance.

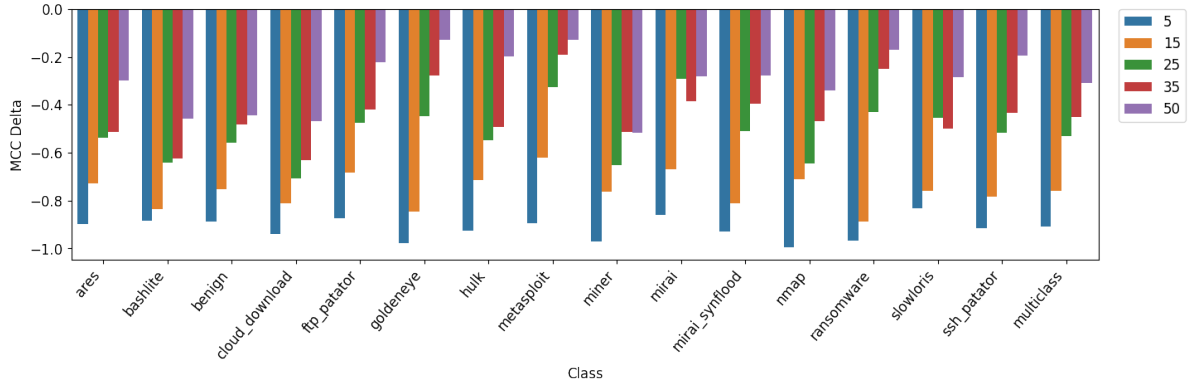
Feature Extraction by Entropy

Feature extraction employing the entropy of the persistence performs significantly worse than all other methods, as evidenced by Figure 32. While we can still observe a clear dependence on window size, the performance is, in general, significantly worse than simple state detection. This implies bad **information retention**, and accordingly, we do not deem this extraction method suitable for this specific problem. The high information loss is to be expected considering the methodology (cf. section 4.2.1), which condenses the lifetime of all topological features into a single value per dimension. Conclusively this implies the high importance of small differences in the data topology rather than a few notable disparities.

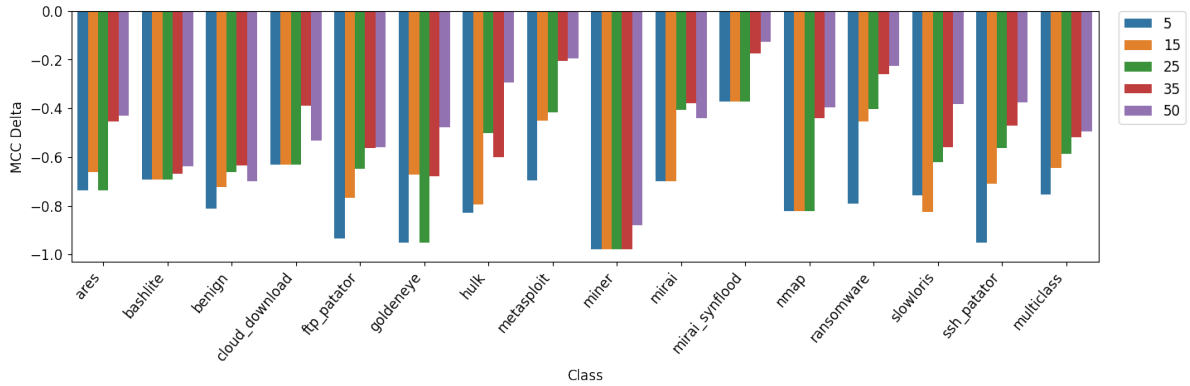
Due to the method's performance, we do not expect good **class separability**. The separability index confirms this, which lies at only 0.64. Interestingly, this is still higher than we can observe on the unprocessed state dataset. This highlights the importance of feature space transformation by preprocessing, as observed during the state dataset analysis.

Feature Extraction by Pairwise Distance

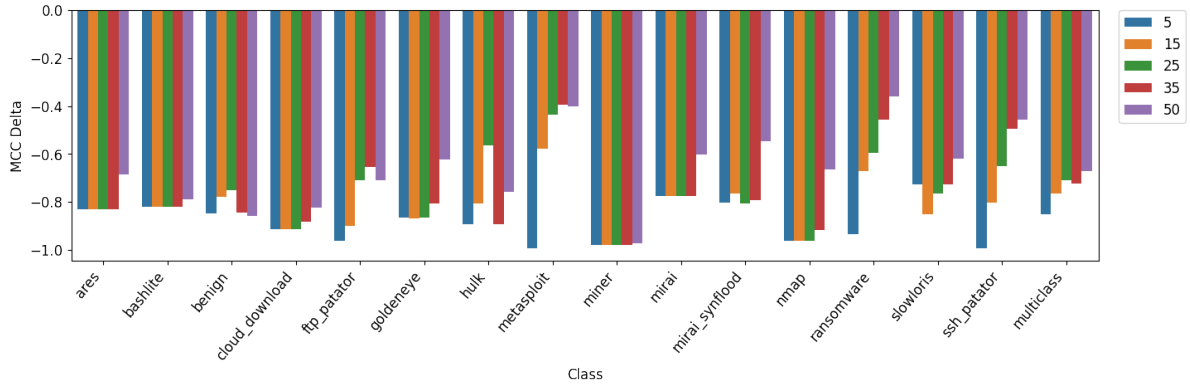
Feature extraction by pairwise distance relies on the distance metric employed. In the case of this scenario, the Wasserstein distance marginally outperforms its Bottleneck counterpart. Regarding **information retention** as shown in Figure 33, we can observe behavior similar to extraction by binning. This means we have a clear correlation between time steps and performance, as well as a correlation between attack type and performance. All in all, the



(a) Random Forest

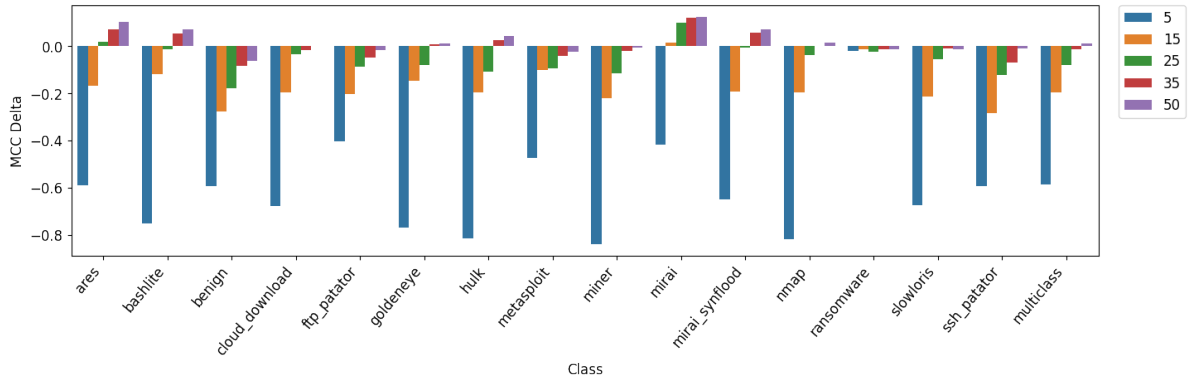


(b) SVM

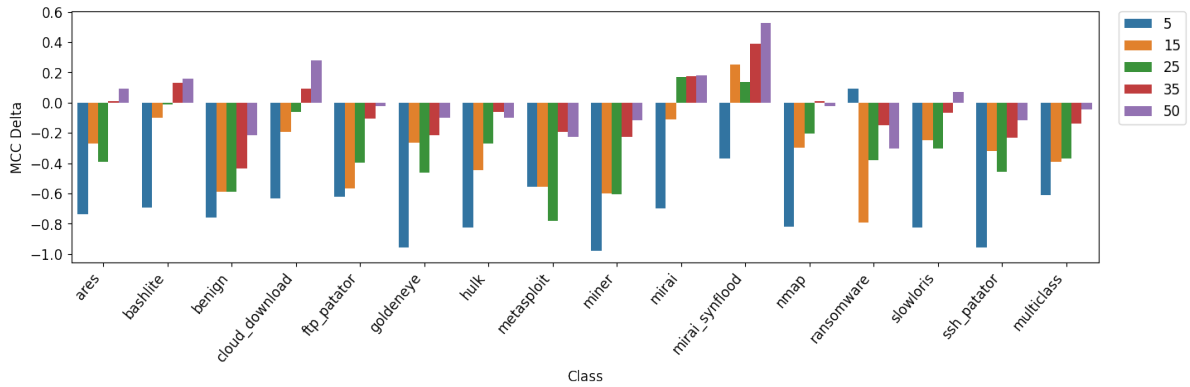


(c) MLP

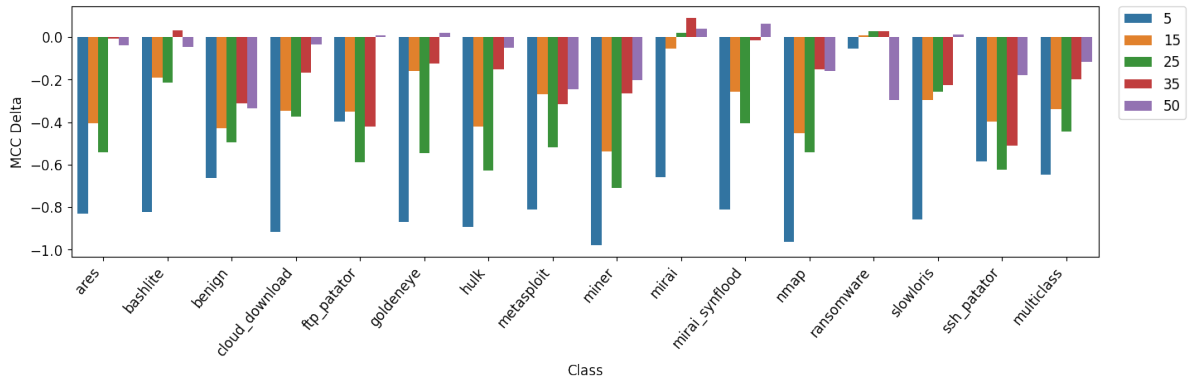
Figure 32: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction by entropy.*



(a) Random Forest



(b) SVM



(c) MLP

Figure 33: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction by pairwise distance (Wasserstein).*

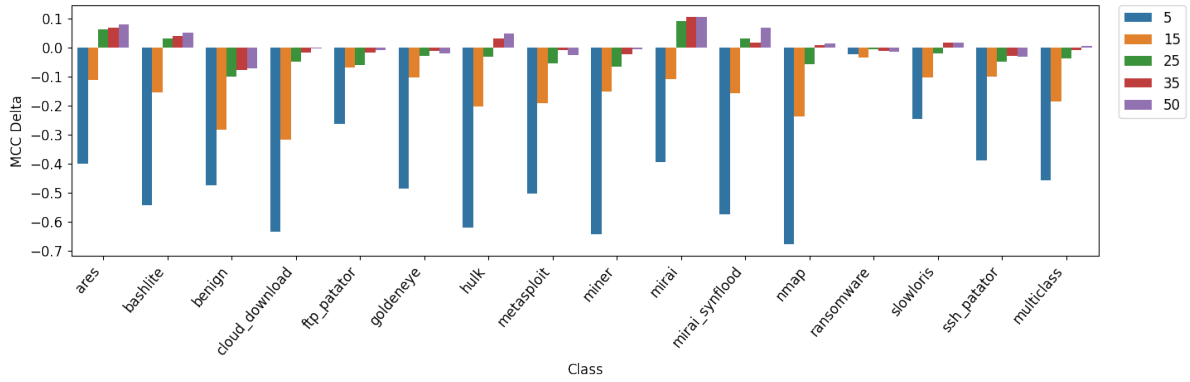
MCC Δ (behavior-analysis MCC minus state-analysis MCC) at 50 time steps is either positive or close to zero, indicating a good feature embedding, albeit slightly less performant than what we have seen from binning. With the generally good performance, we expect good **class separation**, which is confirmed by the separability index of 0.97, which is only slightly below binning's index. In comparison employing Bottleneck as a distance metric, we reach 0.95, explaining the worse performance. This hints at the importance of smaller differences over multiple topological features instead of a clear difference in the greater structure of the data.

Feature Extraction employing Codebooks

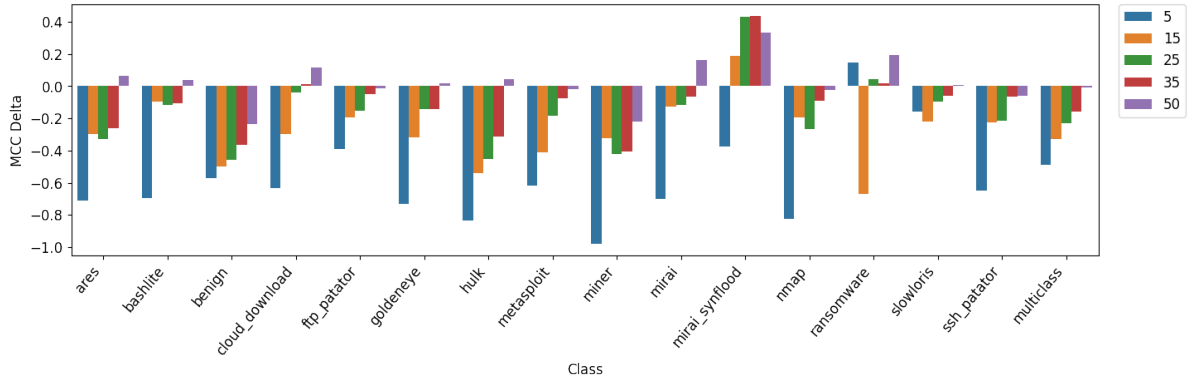
The performance of the feature extraction by codebooks depends on the number of codewords, with 32 performing best in our experiments. We once again graph the MCC Δ as shown in Figure 34. The behavior regarding time steps and attack classes is very similar to the one observed for the other methods. Regarding **information retention**, we can notice parallels to defining quantities with slightly worse performance. In particular, performance on 5 time steps suffers, indicating worse information retention. The separation index nicely visualizes the difference in **class separation** with regard to the number of codewords. With an SI of 0.86, 0.88, and 0.87 for 8, 16, 32, and codewords, respectively, it does not fully correlate with the observed classification performance. This indicates a reliance on preprocessing methods to transform the feature space. In fact, the MCC Δ between none and best performing preprocessing is with 0.03 notable, while the Δ for the most similar performing extraction method — defining quantities — is 0.00.

Assessment

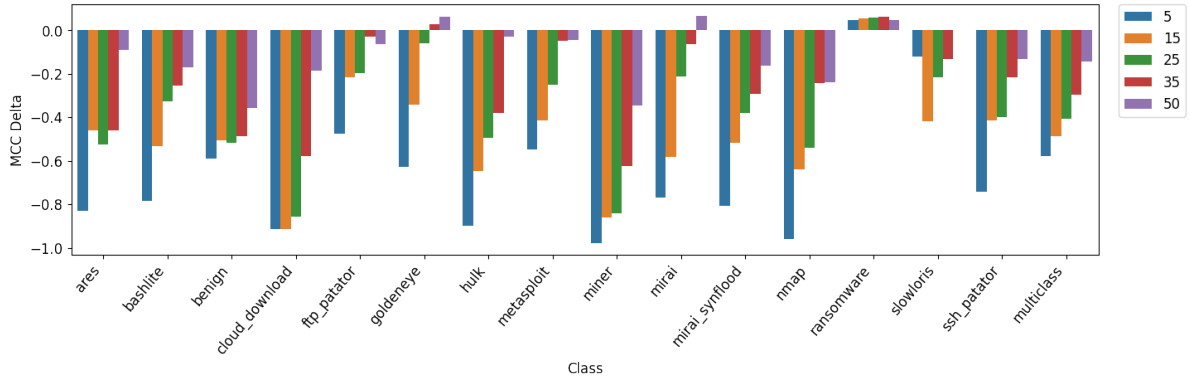
Given the overall better performance compared to the state detection, we show that behavior detection has merit. This is although state detection features to some extent already behavior analysis, in particular in the sensor data fusion of *new_process_info* and *network_activity*, as we utilize TDA to compare the inter-device behavior difference over the aggregation interval. Nonetheless, full behavior analysis still provides benefits, especially regarding the classes most impacted by these sensors. This speaks for sufficient information retention as well as good class separation by at least a subset of the extraction methods. With the exception of entropy-based extraction, we can observe this claim being met. In general, all methods feature good enough information retention and excellent class separation. Especially, the class separation is significantly above what we can achieve with state detection. This allows for simpler classifiers offsetting the cost of the feature extraction. Regarding information



(a) Random Forest



(b) SVM



(c) MLP

Figure 34: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction employing codebooks (32 codewords).*

retention, while sufficient, we can clearly observe substantial enough information loss, with only large time windows allowing for good classification. However, the easier classification can potentially offset this additional computational cost.

There are apparent performance differences between the extraction methods. While for the most part consistent with our assumptions made in section 4.2, there are some methods performing notably worse than expected. In particular, entropy and codebook (with small amounts of codewords) achieve noticeably worse results. This indicates similar distribution characteristics between the lifetimes for every dimension. Consequently, we can conclude topology differences to be minute and thus require more fine-grained methods to capture.

With regard to our pre-formulated hypotheses, we can deduce the following. With respect to **H2** regarding the performance of our sensor set, we once again see merit especially considering the covert attacks. However, to fully evaluate this, we still require a comparison to T-ChangeDrift. More conclusive are the results concerning **H3A**. We clearly show that we are capable of retaining necessary information and adding classification beneficial information regarding the behavior of the devices. We also have shown clear dependence on the number of time steps, which shows that there is valuable data in device behavior. However, while we are able to achieve better performance, the result is less clear than we hypothesized. On the other hand, we also find a high benefit in class separation, which strongly indicates the possibility of using low complexity classifiers and, in return, the capability of offsetting the computational cost of the feature extraction.

6.2.4 Persistent Homology based classification

As a **third stage**, we employ TDA-based classification methods. The particular algorithms utilized are described in section 4.3. Using these methods, we aim to utilize structural properties of the data that traditional classifiers may not be able to pick up on. Due to the nature of our data being sensor values, the topology of the data encodes device-specific behavioral patterns, which are potentially useful for detection. Unfortunately, due to the very high runtime of specifically the TDA autoencoder (over a year for the necessary experiments on our hardware), we cannot fully evaluate its usability for behavior detection. However, we expect only small benefits of utilizing the autoencoder for behavior detection as the same information we aim to add using its batch analysis should be readily provided within the feature vectors by the topology-based extraction.

TDABC

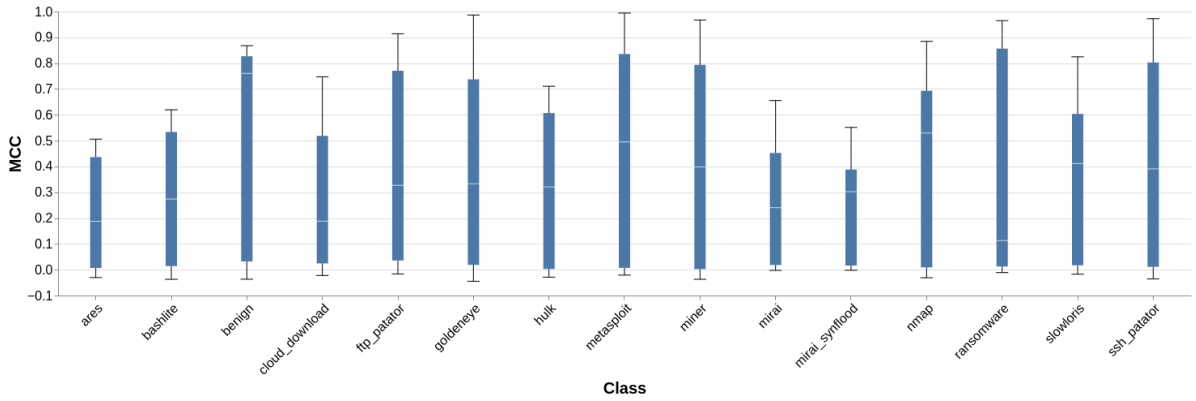


Figure 35: *Boxplots visualizing per class classification performance of the TDABC. Whiskers show min-/max-values.*

TDABC achieves its best performance by employing robust scaling and utilizing the birth time of the longest-lived topological features as filtration distance. In this setup, it reaches an MCC of 0.79. This is significantly worse than what we have observed with the other evaluated classifiers regarding state detection (there SVM comparatively performed worst with an MCC of 0.83). However, given what we discovered regarding class separation, this is to be expected. With an SI of just 0.37 any nearest neighbor approach will struggle. This is also visualized by Figure 35, showing the class-based performance wth the whiskers indicating min-/max-values. Visible is the performance difference between covert and loud attacks. Given the already difficult detection of the covert attacks, the poor class separation leads to insufficient detection, especially for these classes.

Additionally, this behavior is also consistent with our observations regarding the dataset's structure (cf. section 5.5), and the results of the original paper (cf. section 4.3.1). In particular, the classification method handles larger numbers of overlapping classes (as seen in their evaluation with the Swissroll dataset) only poorly. Unfortunately, this class behavior is exactly what we can observe in our use case, especially regarding the covert acting attacks (cf. Figure 19). Thus, this characteristic of the classifiers renders it unfit for state-based detection.

Intriguing is the large span of the boxes' body. Explained is this by the method's heavy reliance on chosen filtration distance. As introduced in section 4.3.1 the original authors propose three methods to set the filtration distance based on the lifetime and birth of the topological features. In particular, these are choosing the birth time of one feature at random, choosing the birth time of the longest-lived feature, or choosing the birth time of the feature

with the lifetime closest to the mean. Regarding the classification performance, the maximum choice outperforms the rest, followed by random, and lastly mean for our data. Importantly, the choice by mean performs only as well as a random classifier would. This gives us insight into the topology of the data. In particular, there appear to be several features not providing significant information regarding the nature of the data instance.

With regard to behavior detection, we expect a performance increase due to the significant increase in class separability. With an MCC of 0.93 it even manages to outperform the MLP. Regarding extraction methods and time steps, we observe the same trends as with the traditional classifiers. In particular, there is a strong correlation between MCC and the number of time steps, with entropy performing significantly worse than the other extraction methods.

TDA-Autoencoder

The autoencoder only allows the differentiation between benign and malicious instances by nature of the algorithm. Hence, we evaluate its performance on a binary version of the dataset, differentiating only between benign and malicious. It achieves its best result of a 0.73 MCC employing min-max normalization, 128 neurons for the first and second layer, and 32, 16, and 4 neurons for the third through fifth layer, respectively. Compared to the traditional classifiers, this is indicates a bad classification performance [1]. However, we have to note that an autoencoder by design is an anomaly detection method. Given the high number of malicious instances behaving very similar to benign instances (all covert acting attacks), it is thus ill-fitted for this classification problem. We originally planned to utilize it to differentiate between different classes, which is, unfortunately, due to time constraints, not possible and might be interesting for future work.

Figure 36 visualizes the impact of preprocessing and outlier percentage choice on precision and recall. As expected, varying the outlier percentage shifts the performance of the other configurations without changes to their relative ordering. The outlier percentage defines how many of the instances are labeled as positive (i.e., malicious) and consequently dictates the number of false positives, resulting in a better (or worse) recall.

Interesting is the influence of preprocessor choice. We can observe a clear trend between choosing no preprocessing and actually preprocessing the data. While certain configurations result in worse performance, we can conclude that all tested preprocessing has merit, with no preprocessing reaching only mediocre to bad results. However, we can not observe the

[1] As the autoencoder performs a binary classification, the respective MCC values can not be directly compared.

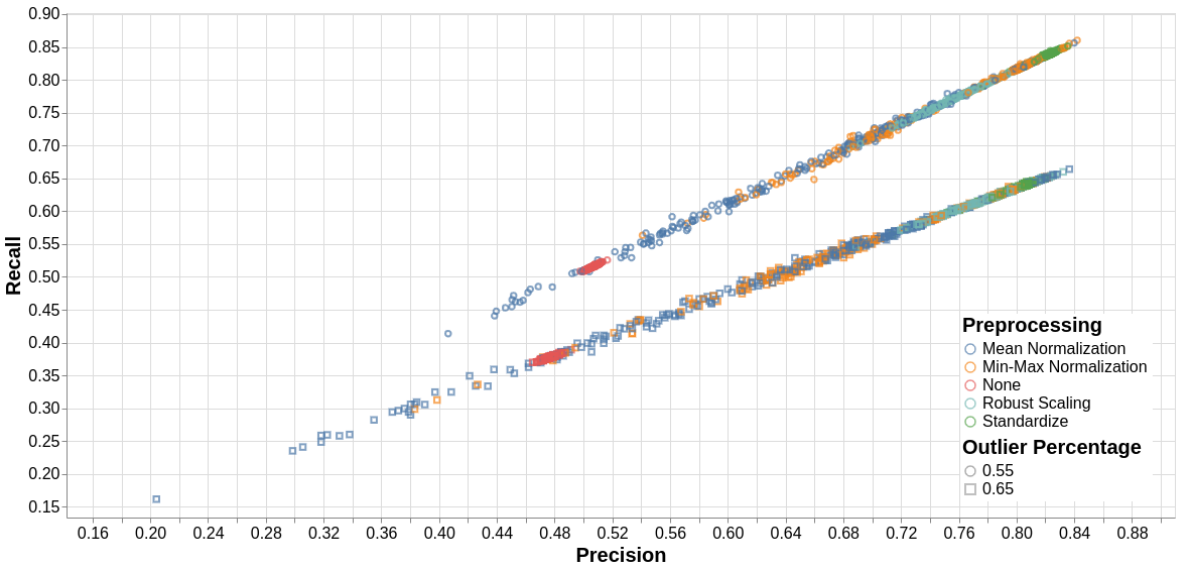


Figure 36: *Precision and recall depending on outlier percentage and preprocessing choice. Each point represents a tested classifier configuration.*

autoencoder strongly favoring any particular preprocessing method (outside of standardization slightly outperforming robust scaling). This is notable as we can observe a clear dependence on the chosen preprocessor in the state detection employing MLP and SVM as a classifier. Thus, it is more akin to the behavior of the Random Forest (though requiring some form of preprocessing) (cf. section 6.2.1). We established in previous sections the presence of scaling issues for a subset of the features, resulting in the need for general preprocessing. However, there we also observed no notable reliance on strict anomaly preserving scaling methods to retain the anomalous characters of the respective instances, thus explaining this behavior.

Furthermore, we can not detect any correlation of the MCC with the batch size. As discussed in section 4.3.2 the TDA-Autoencoder utilizes each mini-batch to regularize the reconstruction loss based on differences in topology between input and latent space. We discovered during the analysis of behavior detection that device behavior (i.e., sensor states over multiple time steps) carries not only a significant amount of information but is capturable, employing persistent homology in combination with persistence distance metrics (as utilized here). Thus, we can derive that the absence of a correlation between MCC and batch size is most likely induced by little information gain after a batch size of 32 and not the inability to extract meaningful information. This matches our observations regarding the pairwise distance feature extraction, showing notable more minor performance improvements going from 35 to 50 time steps than for smaller window sizes.

Unfortunately, due to time constraints (running the necessary experiments on the hardware at our disposal would take roughly a year), we cannot evaluate this classifier's performance for behavior detection. The TDA-based feature extraction is able to strengthen the anomalous character of even covert attacks, as we show in section 6.2.3. As the autoencoder relies on the instances' anomalous nature, this should benefit the classification. However, the persistent homology based regularization should have already introduced a significant amount of the information we make available to traditional classifiers, employing TDA-based feature extraction. Thus, while we expect an improvement, it should not be as notable as with the TDABC algorithm.

6.2.5 TDA-based behavior comparison in contrast to traditional approaches

T-ChangeDrift is an intrusion detection system designed for homogeneous IoT devices during a previous master's project. All functional details we may discuss in the following refer to Tennié's original work [Ten21]. T-ChangeDrift is a complete system, meaning other than our work here, which is primarily research-focused with consequently the goal of data collection and evaluation, Tennié provides a fully deployable pipeline. As such, there are a number of parts that have no relevance to us. In particular, we are concerned with device behavior and behavior comparison.

To achieve this, Tennié employs hierarchical agglomerative clustering on the device's behavior as given by the sensor data. The result is a cluster tree for every single device representing its past behavior. Each node is represented by the union of its device states. For numerical values, the union is given as the average; for sets, we can compute the actual union, and for lists, the union represents the concatenation of all lists' values. This allows for the calculation of tree distances. In particular, T-ChangeDrift employs the tree alignment distance (TAD). To calculate the TAD, the two trees are aligned in such a way that minimizes the sum of the distances between every matched node pair [Bil05]. The distance is given by a cost function, which in our case, calculates the sensor difference as discussed in section 5.5. The minimum cost sum is the tree alignment distance.

Given this procedure, T-ChangeDrift's detection method can be classified as a structural detection, as it searches for anomalies in data topology. The problem we see with this approach is the reliance on hierarchical agglomerative clustering and, therefore, the exclusive consideration of zero-dimensional topological features. We believe that the information loss is significant enough to warrant the examination of higher dimensional topological features as done in our behavior analysis. We formulate this claim in **H3B**.

In the following, we evaluate the performance of T-ChangeDrift's behavior feature extraction versus our own. To this end, we utilize the T-ChangeDrift configuration found to work best in Tennié's work. Specifically, this is the usage of the Manhattan distance over the Euclidean as well as complete linkage for the clustering process. We complement this with our own findings on this specific dataset, primarily the use of 50 time steps. To not contaminate the results of T-ChangeDrift, we omit the sensors *new_process_info* and *network_activity*, as otherwise we would be forced to perform TDA-based behavior comparison on the aggregation-level scale to be able to measure the distance between the sensor values. As we are only concerned with the extraction of device behavior features, we do not employ T-ChangeDrifts DBSCAN classification and instead rely on our classifiers to facilitate a comparison that allows the attribution of a performance difference to the actual feature extraction process.

Assessment

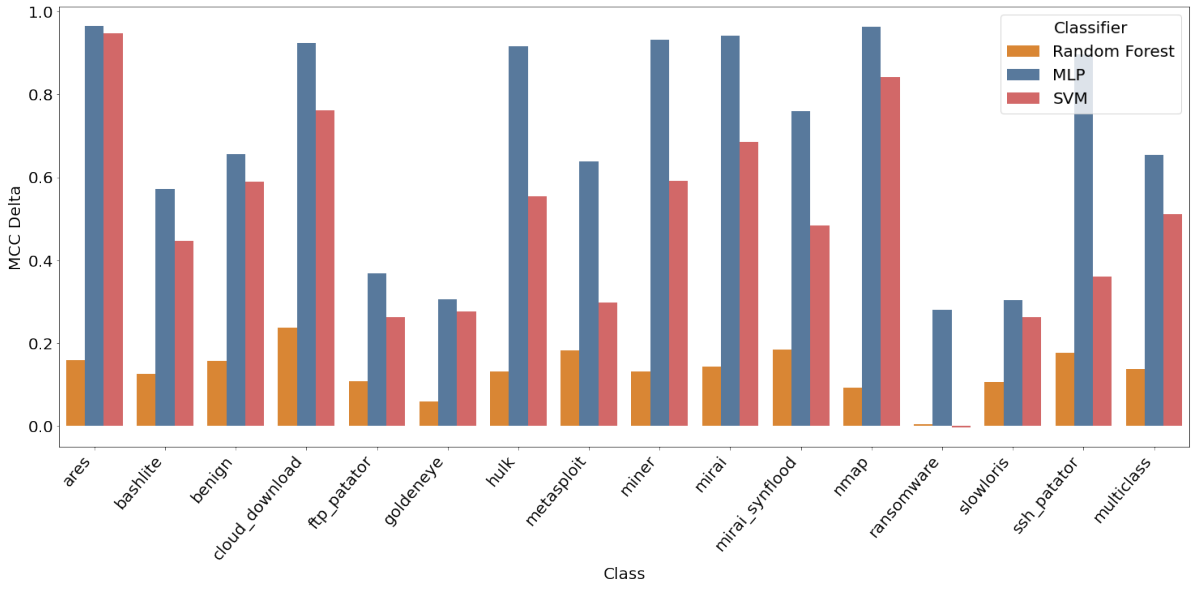


Figure 37: *Difference in MCC per class depending on classifier choice between T-ChangeDrift and TDA extracted behavior (our MCC minus T-ChangeDrift's MCC). In both cases, the best performing classifier configuration is selected.*

Figure 37 visualizes the MCC difference between the results achieved employing our approach and T-ChangeDrift (our MCC minus T-ChangeDrift's MCC). It is clearly visible that we outperform T-ChangeDrift regardless of classifier choice and in all classes. We can not observe any noteworthy correlation between class and performance Δ. Consequently, TDA-based feature extraction captures higher amounts of information regardless of how noisy the attack

is. The multiclass MCC Δ is 0.65 for the MLP, 0.51 for SVM and 0.14 for Random Forest. While all of them are quite substantial, there is a clear difference between SVM and MLP on one side and Random Forest on the other. This indicates suboptimal hyperparameter choices for the T-ChangeDrift produced embedding. Random Forest on the other hand is largely self correcting regarding feature scale and weighting. Accordingly, the performance difference observed when employing the Random Forest classifier should be considered the main indicator of difference between the embeddings methods.

T-ChangeDrift produces data with very clear anomalies and thus performs for both SVM and MLP best with robust scaling. For our approach, no preprocessing is the best method for MLP, and min-max scaling for SVM indicates the absence of clear outliers. As Random Forest is very robust to feature scale, both T-ChangeDrift and our approach show no clear preferences for a specific preprocessing method.

All in all, our results show that **H3B** firmly holds. Utilizing topological data analysis for information extraction instead of relying on hierarchical agglomerative clustering and tree comparison results in significantly better feature vectors. We strongly outperform T-ChangeDrift regardless of classifier and class (with the exception of ransomware and SVM). Consequently, we can conclude that we are able to capture relevant information that T-ChangeDrift can not.

6.3 Distributed Intrusion Detection

In the second scenario we consider a setup with a distributed intrusion detection process (cf. section 5.2 and section 5.5.2). This entails the distribution and processing of sensor data on all devices. Intuitively the process introduces a significant amount of noise that we predict will impact sensors monitoring network and file access quite heavily. We employ the same attack and sensor setup as employed for the first scenario (cf. section 6.2).

The evaluation follows the same schema as the one for the first scenario. We evaluate the merit of TDA at various stages of the detection utilizing the classification procedure given in Figure 23. We assess sensor importance and compare TDA-based extraction with T-ChangeDrift's approach. Additionally, during the entire evaluation, we also compare results with the ones achieved during the first scenario to assess the impact of noise on both classification in general and TDA-based extraction in particular.

6.3.1 Anomalous State Detection

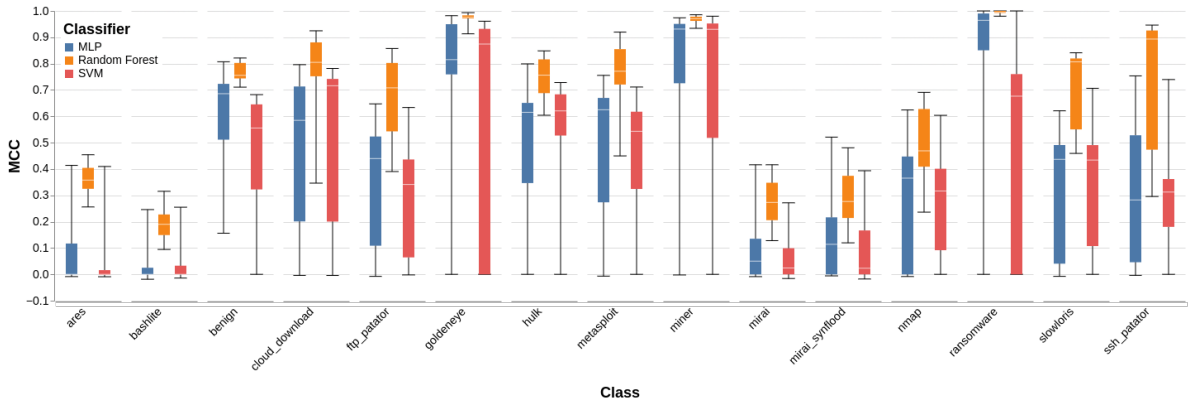


Figure 38: *Boxplots visualizing the per class classification performance per classifier given by the MCC. Whiskers show min-/max-values.*

We now consider the benefit of TDA at **stage 1** when presented with a noisier input. As anomalous state detection is only concerned with the current system state, we predict a high impact of the noise introduced by the distributed intrusion detection in comparison to what we have seen in scenario one. Otherwise, we expect the same trends with regard to the best performing classifier and class performance. However, we assume less impact of the scenario on classes (predominantly) not utilizing the device's network (e.g., ransomware, miner).

Figure 38 shows the per class performance (as given by the MCC) of the different classifiers as boxplots, with the whiskers visualizing min- and max-values. The impact of the IDS noise is clearly visible. Especially, the primarily covert acting botnets are significantly harder to detect. As discussed earlier, this is to be expected, as these classes are already behaving very similar to benign instances, with the introduced noise only blurring the line further. Consequently, we can also observe worse performance for the detection of benign behavior. Interesting is the clear performance hit on network-focused attacks (such as Nmap, Cloud Download, or DDoS (Slowloris, Hulk, Synflood)) while the detection of attacks predominantly not utilizing the network (such as Ransomware and Miner) is almost identical to the one in scenario one. This confirms our speculation about how the noise — due to its nature — will impact the class detection differently.

On the other hand, the general trends are very similar to what we observed in the first scenario. Random Forest once again outperforms all other classifiers, with all three showing apparent weaknesses in detecting covert acting attacks. Random Forest achieves a maximum MCC of 0.77 (Δ0.20 compared to scenario one) employing no preprocessing and 128 trees.

MLP once again performs second best with an MCC of 0.6605 (Δ0.2671 compared to scenario one) while utilizing robust scaling, a learning rate of $\sim$ 0.1731 and 32 hidden units. The worst performance with an MCC of 0.58 (Δ0.25 compared to scenario one) is reached by SVM, employing a γ of $\sim$ 0.2798 and a C of 1.8616 as well as scaling by mean. Accordingly, the best performing classifiers employ preprocessing and hyperparameters similar to the ones in scenario one, although with significantly worse results. The behavior can be explained due to the utilization of the same devices, sensors, and attacks.

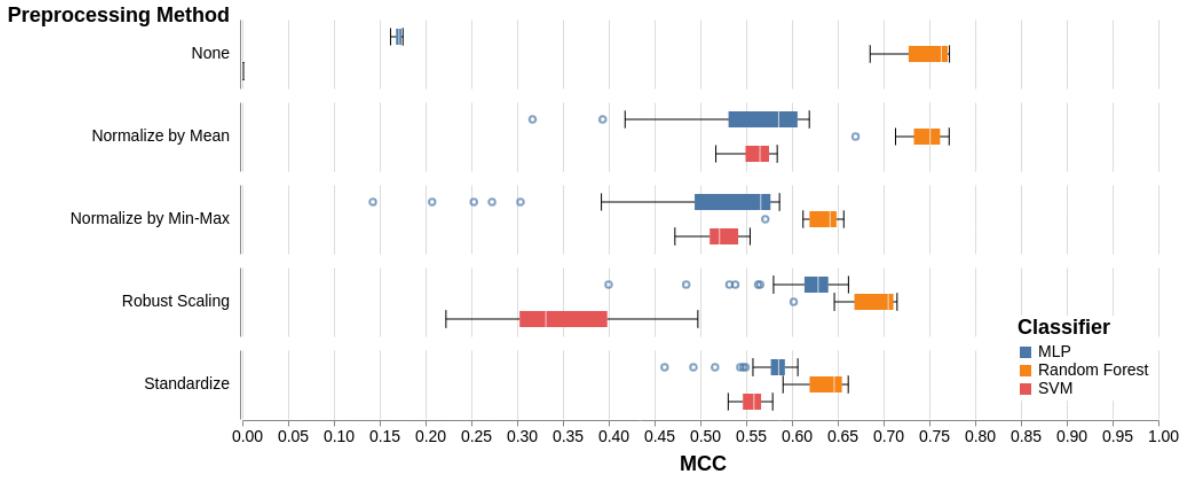


Figure 39: *Boxplots visualizing the impact of preprocessing choice on classification performance. Whiskers show upper and lower quartile.*

On a closer look at the preprocessing methods, we can observe close to identically behavior as visualized in Figure 39. The figure visualizes the worse performance but also shows the clear reliance of MLP on robust scaling or standardization, while SVM favors scaling by mean or standardization. Random Forest — by nature — is again largely indifferent to preprocessing, with none leading to the best results. We can also once again observe the scaling issues of the sensors *new_process_info* and *network_activity*, resulting in the incapability of meaningful classification for both MLP and SVM if no preprocessing is applied (cf. section 6.2.2).

Assessment

Disregarding the overall worse performance, we can observe very similar trends to scenario one. This once again validates our hypotheses, particularly regarding attack class impact. With respect to **H1**, we still consider it as met, as the performance — while worse — is still decent and within the margin that we expected. However, we will discuss this in more detail

in the following section, focusing on the sensor importance and specifically the impact of the IDS-founded noise.

6.3.2 Sensor Importance

As with the sensor importance in the prior scenario, we focus on the TDA-enabled sensors, namely *new_process_info* and *network_activity*. In addition, we evaluate the impact of the distributed IDS and, consequently, the added noise on the sensor importance itself. This then allows us to establish a definitive answer regarding the question of the sensor selection's merit raised in hypothesis one. To achieve this, we first focus on general sensor importance and afterward dive deeper into the TDA-based sensor data fusion (**stage 1**).

General Sensor Importance

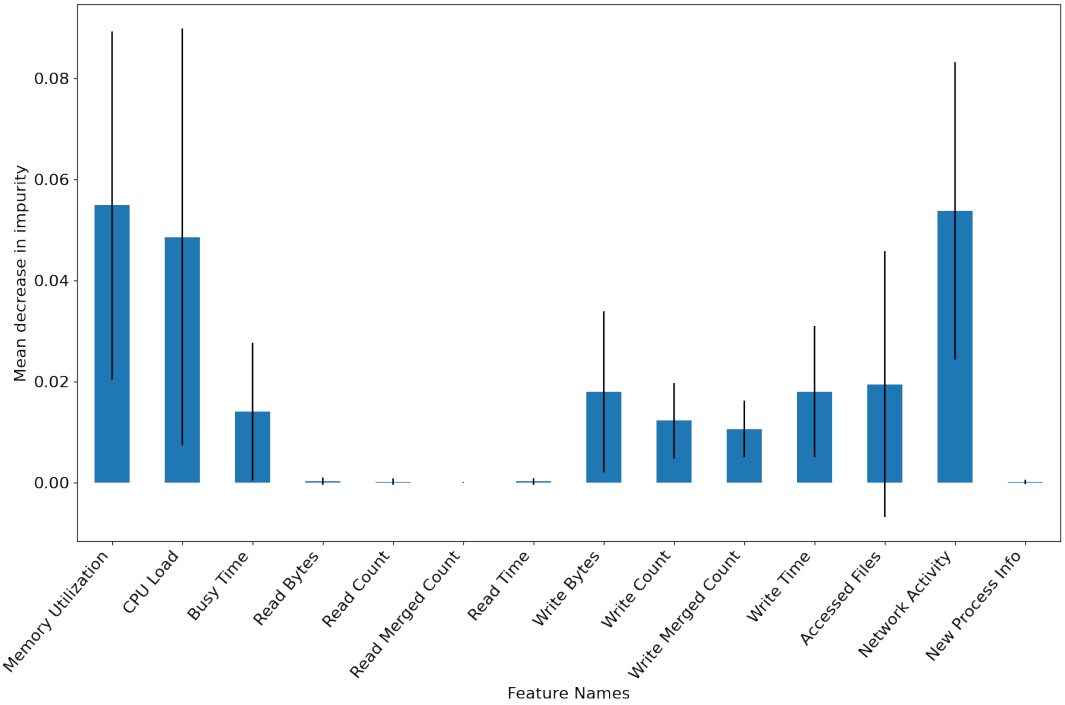
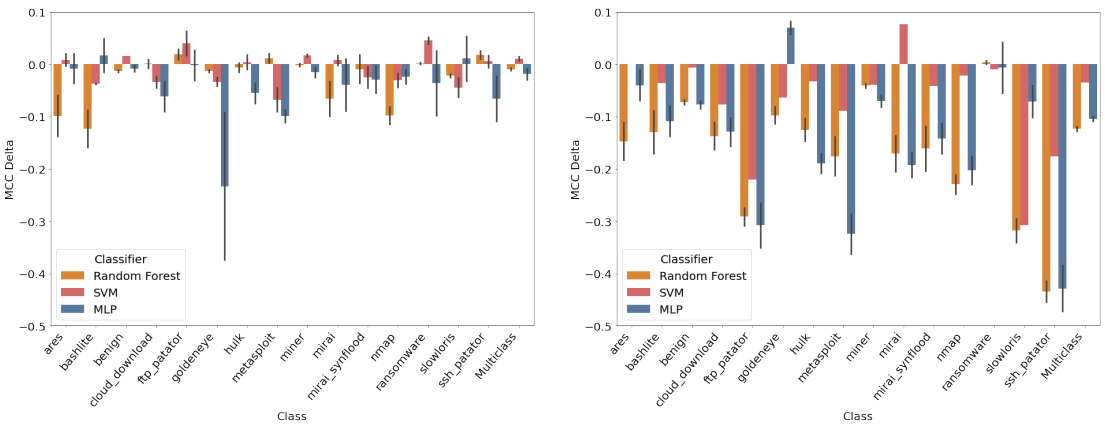


Figure 40: *Barplot showing the mean decrease in impurity per feature as reported by the best performing Random Forest model (errorbar shows standard deviation) [Ped+11].*

We utilize the best performing Random Forest classifier setup to quantify feature importance by mean decrease in impurity. The resulting graph is given in Figure 40. In direct comparison to scenario one, as given in Figure 27, we can observe an almost perfect match with regard

to feature importance ranking. However, there are significant differences in the impurity reduction. It is important to note that we can not directly compare the values, as they are subject to the dataset. The noise caused by the distributed IDS impacts the expressive capabilities of each individual feature resulting in a lower possible impurity reduction for each. Nevertheless, we can compare the relation between the feature-specific impurity reduction. Here the type of noise becomes noticeable. Most notable is the relative increase in "CPU Load" importance. It now outperforms "Accessed Files" and matches "Network Activity", which holds significantly more importance in the scenario. These two are the sensors that we expect to be impacted most heavily, as the distributed IDS results in heavy network and disk usage. Interesting is also the relative decrease in the importance of "Memory Utilization". The other devices' sensor values are temporarily being held in memory and thus impact this sensor as well.

Sensor Importance and Information retention of TDA-based sensor data fusion



(a) Feature: *new_process_info*

(b) Feature: *network_activity*

Figure 41: *Barplots visualizing the average MCC difference between the dataset with and without specific sensors enabled (MCC with sensor - MCC without sensor).*

We evaluate the sensor importance of *new_process_info* and *network_activity* by omitting the respective sensor and assessing the MCC in direct comparison. Please refer to section 6.2.2 for further details. The resulting MCC Δ (MCC with sensor - MCC without sensor) is given as barplots in Figure 41. The experiments are conducted ten times each. Accordingly, the bars showcase the mean MCC, with the standard deviation visualized as error bars. Note that we discovered scaling problems that impact the Kernel SVM (cf. section 6.2.2), which is also the case for these results.

While we can observe similar global trends (*network_activity* holding more value than *new_process_info*), we can easily discern the influence of the changed intrusion detection approach. The noise introduced by the detection process first and foremost influences network and disk activity. However, the processing of the other devices' data influences all of the sensors, with the important difference that especially the simpler sensors (e.g., CPU load) do not contain enough information to differentiate between noise and anomaly. The *network_activity* — although heavily impacted — can provide this differentiation (e.g., utilizing the distinct destination port 13024 employed by the IDS). Consequently, it holds significantly more importance than in scenario one. This is nicely visible in Figure 41(b). Also, note the significantly higher decrease in MCC compared to scenario one. Interesting is the class-specific impact. As expected, the features carry little value regarding attacks primarily not dependent on network traffic (cf. Ransomware and Miner). However, we can observe a high impact for classes previously easily distinguishable by the sheer number of connections alone. Namely, these are Bruteforce and DDoS attacks. Notable exceptions are Hulk and GoldenEye, both establishing enough connections to put a clearly measurably strain on the victim's computational resources, thus detectable by other sensors as well.

The *new_process_info* sensors appears to no longer provide clear beneficial information. Note in particular the high variance. We can still see merit with regard to certain classifiers and specific classes, like the botnet attacks. However, overall its value is clearly diminished in direct comparison to the first scenario. This is surprising as this sensor should be robust against the type of noise added by the intrusion detection process

Assessment

With the introduction of noise, our *new_process_info* sensor looses its value. Coupled with its high computational expense, we thus conclude that it does not provide enough benefits to merit its existence. With regard to our hypotheses, we thus conclude the following. **H1** strongly holds in both scenarios. TDA-based sensor fusion is capable of extracting the difference between complex sensor values of two devices. However, with respect to **H2**, only *network_activity* proves its value over both scenarios. Nonetheless, we still believe that a sensor monitoring process behavior has its place in the defined scenario. Thus, we encourage potential future work investigating how the information provided by such a sensor can be increased while simultaneously decreasing its computational cost.

6.3.3 Anomalous Behavior Detection

Stage 2 — anomalous behavior detection — analyses device behavior over a period of time instead of focusing on a device state at a single point in time. This is particularly interesting for the current scenario, as it can improve the noise robustness. Additionally, we already observed both excellent class separation and information retention when employing device behavior in our evaluation of scenario one. Once again TDA-based feature extraction is utilized to generate feature vectors describing the devices' behavior. Refer to section 4.2 for detailed explanation and motivation of the methods. The main focus of the following is the evaluation of the impact of time window length as well as the impact of the different extraction methods. Accordingly, given Figure 24 we consider the path stage 1, into stage 2, into classification. Additionally, we consider key differences to observations made in section 6.2.3.

General Performance

Table 10: *The table shows the best MCC achieved per class depending on different classifiers.*

Time steps	Ares	Bashlite	Benign	Cloud Download	FTP Patator	Goldeneye	Hulk	Metasploit	Miner	Mirai	Mirai Synflood	Nmap	Ransomware	Slowloris	SSH Patator
MLP															
5	0.00	0.37	0.00	0.94	0.00	0.00	0.53	0.65	0.00	0.44	0.60	0.00	0.14	0.00	0.09
15	0.58	0.59	0.61	0.97	0.56	0.63	0.74	0.67	0.40	0.74	0.89	0.70	0.68	0.49	0.64
25	0.65	0.71	0.69	0.88	0.74	0.70	0.71	0.67	0.69	0.66	0.79	0.72	0.82	0.78	0.78
35	0.86	0.80	0.86	0.96	0.88	0.89	0.86	0.84	0.83	0.79	0.86	0.83	0.91	0.89	0.89
50	0.94	0.93	0.96	0.99	0.95	0.96	0.95	0.96	0.93	0.96	0.97	0.98	0.99	0.96	0.96
Random Forest															
5	0.19	0.39	0.28	0.94	0.21	0.33	0.62	0.51	0.16	0.54	0.65	0.37	0.38	0.36	0.40
15	0.73	0.77	0.87	0.99	0.75	0.87	0.92	0.86	0.69	0.92	0.93	0.83	0.87	0.72	0.82
25	0.96	0.90	0.97	1.00	0.93	0.91	0.96	0.95	0.94	0.97	0.97	0.95	0.96	0.96	0.96
35	0.96	0.91	0.97	0.99	0.96	0.96	0.98	0.97	0.98	0.96	0.97	0.98	0.98	0.98	0.98
50	0.99	0.95	0.99	1.00	0.99	0.99	1.00	1.00	0.99	0.97	0.99	0.98	1.00	0.99	0.99
SVM															
5	0.00	0.35	0.00	0.97	0.00	0.00	0.44	0.61	0.00	0.33	0.57	0.00	0.20	0.00	0.05
15	0.20	0.49	0.55	0.89	0.32	0.40	0.83	0.63	0.10	0.84	0.85	0.46	0.58	0.18	0.48
25	0.76	0.70	0.85	0.88	0.77	0.81	0.86	0.73	0.83	0.82	0.88	0.82	0.86	0.87	0.84
35	0.92	0.81	0.91	0.97	0.95	0.94	0.95	0.91	0.92	0.89	0.97	0.94	0.98	0.93	0.93
50	0.96	0.85	0.96	1.00	0.97	0.94	0.99	0.97	0.96	0.96	0.98	0.98	0.99	0.93	0.98

Parallel to scenario one, we evaluate a total of 5 time windows length. Specifically, these contain 5, 15, 25, 35 and 50 time steps. For each time window length, we test the entire

classification procedure as detailed in section 5.6. Table 10 visualizes the MCC of the best classification procedure per class per classifier with regard to the number of time steps. For all classifiers, the best performance is achieved when utilizing the largest number of time steps (i.e., 50). This follows what we observe in the first scenario.

Best classification performance with an MCC of 0.99 is reached by the Random Forest classifier. Following are the MLP with 0.96 and the SVM with 0.95. These results are significantly higher than what we reached by employing simple state detection. In fact, they are on par with what we observed in the first scenario. Thus, our assumption regarding the behavior analysis's greater robustness to noise holds true. Behavior analysis is able to neutralize the IDS-induced noise in the data completely.

Note the strong correlation between the number of time steps and MCC. This indicates good information retrieval by the feature extraction methods. We can observe a shift with regard to the best performing extraction methods. While for scenario one — in particular for more extended time periods — binning and defining quantities generate the best feature vectors, now, defining quantities is replaced by pairwise distance-based extraction (specifically utilizing the Wasserstein metric). This is a good indication that deciding differences between the topology of the different classes' point clouds are more minute, thus benefitting the pairwise distance approach over defining quantities. We assess this in more detail in the following section. Regarding preprocessing, we can again observe a weak positive correlation between anomaly preserving methods and MCC; however, nothing particularly noteworthy.

Comparing the differences in class performance between scenarios one and two (cf. Table 10 and Table 9), we can clearly observe the impact of the IDS noise on the behavior analysis. As discussed during the state analysis evaluation, the noise primarily impacts classes heavily relying on network traffic (such as bruteforce and DDoS attacks). The comparison between the two tables nicely visualizes the effect. While for scenario one, even a low number of time steps are sufficient to detect these kinds of attacks, at least 25 time steps are required in scenario two. Importantly, we can again observe the behavior analysis counteracting the noise, with a performance at 50 time steps for both bruteforce and DDoS attacks closely matching the one in scenario one. Additionally, there is still a lower classification rate — particularly at smaller time windows — for covert attacks, as noted in scenario one.

The extraction dependant classification performance with respect to the classifiers is visualized in Figure 42. It showcases the generally better performance of the Random Forest classifiers, as also shown in Table 10. The extraction methods rank almost identically based on

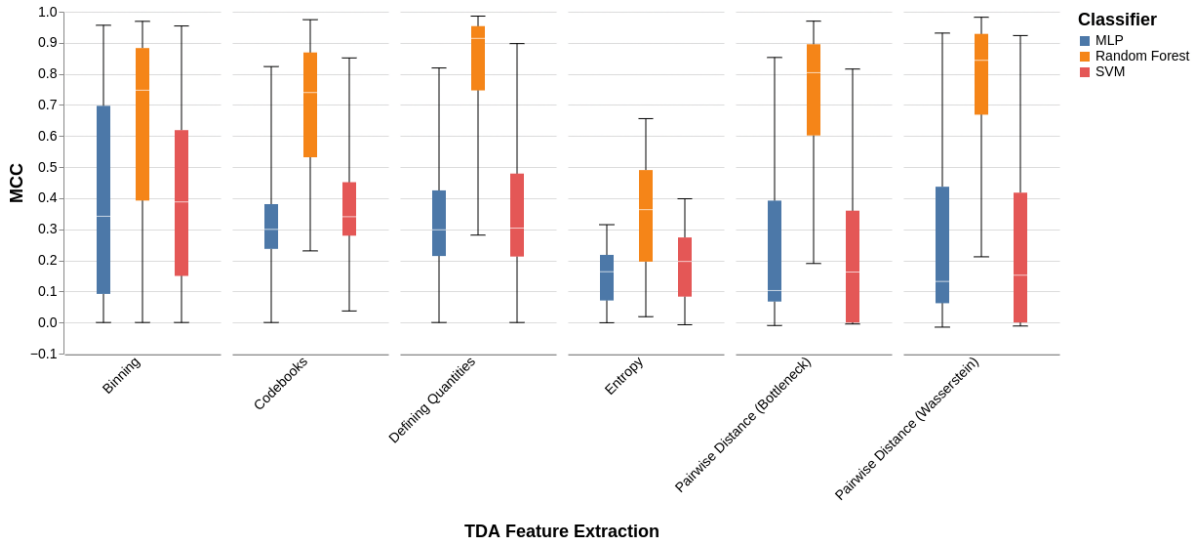


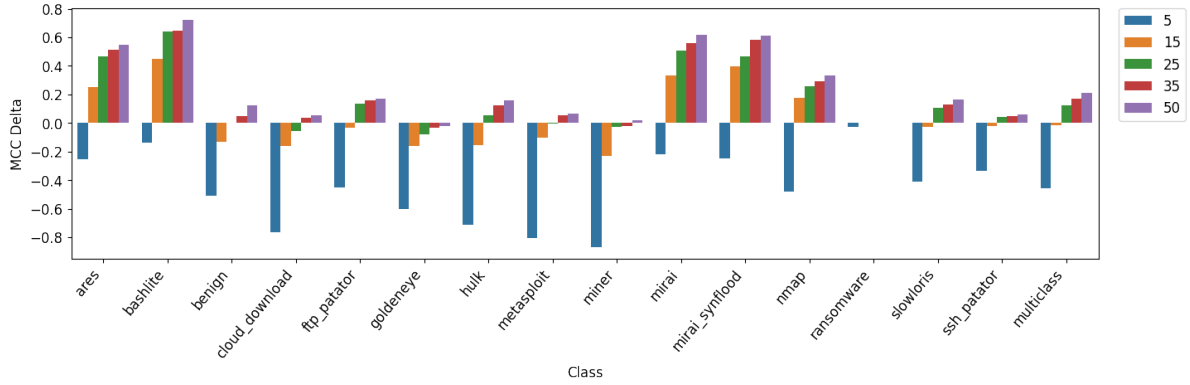
Figure 42: *Boxplots visualizing classification performance (by MCC) depending on feature extraction and classifier choice. Whiskers show respective min-/max-values.*

their MCC as in scenario one. The only key difference is the aforementioned slightly better performance of the pairwise distance based extraction.

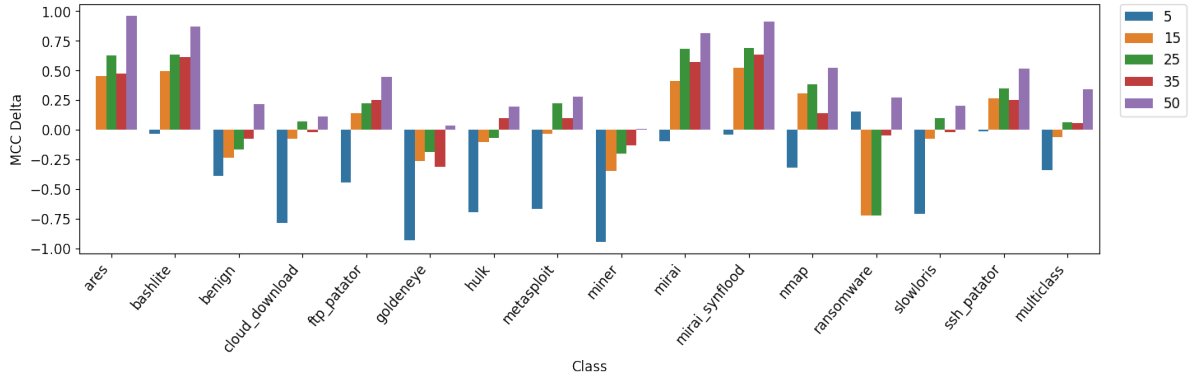
Discussion of Extraction-Specific Performance

All extraction methods achieve very similar results as in scenario one. The overall structure of the datasets is very comparable (same sensor, attacks, and measurement periods) with only the added simulated IDS communication differentiating the two; thus, this is to be anticipated. Nonetheless, there are a few key differences that we want to highlight and discuss.

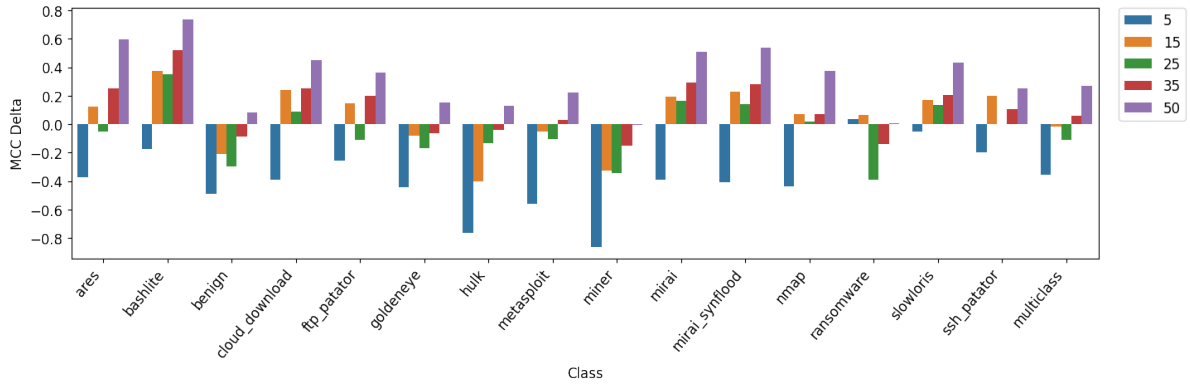
Figure 43 showcases the performance of the pairwise distance (Wasserstein) based extraction depending on time steps and classifier choice (MCC of best performing behavior classifier minus best performing state classifier). Note in particular the time step dependent performance. With scenario one, we only ever observed a meaningful benefit of behavior analysis over state analyses at a minimum window size of 35 independent of the chosen method. Now a window size of 15 is sufficient. This is consistent with all extraction methods on this dataset and highlights very clearly what we already discussed in the general performance evaluation. Namely, the resilience to noise caused by evaluation of multiple devices states, which allows the identification of the noise as due to it being a constant factor. Additionally, it is capable of picking up finer details, as we have shown in the evaluation of scenario one. Quantifiable is this characteristic by considering the **class separability**.



(a) Random Forest



(b) SVM



(c) MLP

Figure 43: *Barplots showing the per class MCC Δ (behavior-analysis MCC minus state-analysis MCC) of all evaluated time steps in comparison to state detection utilizing feature extraction by pairwise distance (Wasserstein).*

We consider the distance-based extraction to illustrate this behavior. However, with the exception of entropy-based extraction, it is also observable for all other methods. The separability index measures 0.41, 0.76, 0.76, 0.80, and 0.94 for 5, 15, 25, 35 and 50 time steps respectively. This clearly shows the progression in noise reduction, with the separability index at 50 time steps closely matching the one in scenario one. To understand this, we have to discuss how noise impacts the topology of the data and, accordingly, the TDA-based extraction.

Topological data analysis is concerned with topological features. Therefore, the derived features are very robust against the type of noise introduced by our IDS simulation. Intuitively this is visualized in Figure 11, which we discussed in our introduction to TDA and persistent homology. The data points in this example are sampled from a circle with a substantial amount of added noise. However, when considering the persistence diagram in Figure 11(b), only the defining feature (i.e., the underlying circle) has a significant lifetime. All noise generated features are short-lived. This lifetime is what we extract in the feature generation. Specifically, the methods are — in different ways — encapsulating alteration to the topology and their significance in a single feature vector. Accordingly, alterations to key topological structures (with consequently long lifetimes) hold substantial weight, while noise is largely ignored.

As each extraction method performs this encapsulation differently, some will be better fitted to a given problem than others. In this scenario, pairwise distance extraction employing the Wasserstein metric performs notably better than in scenario one. For the distance extraction, we calculate the difference between two persistence diagrams as described in section 4.2.3. Here the two diagrams are aligned, and afterward, the distance between matched pairs of points is considered. By definition, this assigns a larger weight to long-lived features as there can be a greater difference in lifetime between the two diagrams. Given the noise in our data, this relative weight is favorable for our specific scenario. The preference towards Wasserstein over Bottleneck can be explained by the differences in general topology between attack and benign instances. As discussed in section 6.2.3, there is not a singular anomalous topological feature signaling the presence of an attack but rather key differences between several defining features. The Wasserstein metric better captures this, as Bottleneck only considers the maximum distance.

The noise reduction effect is thus inherent to the TDA-based extraction but still correlated to the number of time steps. A reduction of device behavior into a single feature vector entirely based on its topology results in information loss concerning the devices' states. In the primarily noiseless environment of scenario one, this loss is outweighed by the information gain at around 35 time steps. The fact that we can now detect a net benefit over the state

analysis at 15 – 25 time steps is due to the added noise reduction. Hence, we can conclude that behavior analysis is highly recommended for any scenario where we can expect moderate to heavy amounts of noise impacting the sensors.

Assessment

TDA-facilitated behavior analysis proves to be highly beneficial for a scenario featuring noise impacted sensor measurements. We are able to observe a significant classification improvement with time-windows as little as 15 or 25 time steps (subject to extraction methods). This is due to good information retention (as discussed in scenario one), but more so thanks to the characteristic robustness to noise of TDA-based extraction. Also, note the high separability index, especially at larger time windows. Once again, this speaks for the possibility of utilizing less complex classifiers. This can offset the feature extraction cost, particularly as we only have to consider fewer time steps than in scenario one.

Given the results obtained here in combination with the ones from the first scenario, we can decisively deduce that our hypothesis **H3A** holds. TDA-enables behavior analysis provides significant benefits for the task of intrusion detection given a set of homogeneous devices. Both in close to optimal use case, but also if presented with noisy sensor data.

6.3.4 Persistent Homology based classification

Parallel to scenario one we evaluate if TDA-based classification methods (**stage 3**), utilizing the structure of the data, can provide a benefit to our use case (cf. section 5.2). In particular, with this scenario, we are interested in potential noise resilience by the respective classifiers. Unfortunately, we can not provide in-depth results with respect to behavior analysis, as the runtime of the experiments (specifically the TDA autoencoder) far exceeded the timeframe of this master's thesis.

TDABC

As observed with all other classifiers, TDABC's performance with regard to state detection is substantially worse. Its best performance, an MCC of 0.59, is reached by employing robust scaling and the birth time of the longest-lived topological feature as the filtration threshold. While still significantly worse than Random Forest's performance, it is on par with the MLP. This is explainable by the SI of 0.62, which is notably better than in scenario one.

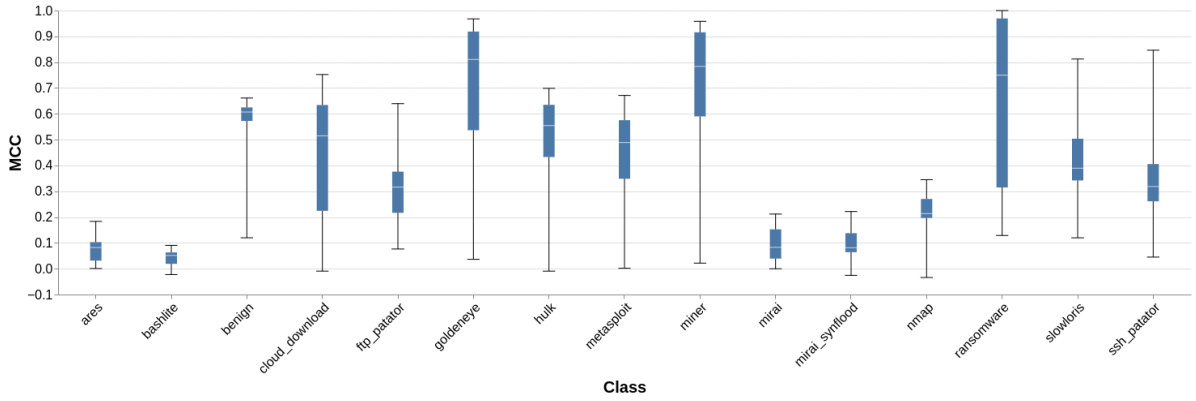


Figure 44: *Boxplots visualizing per class classification performance of the TDABC. Whiskers show min-/max-values.*

Figure 44 gives the class classification performance. We can observe the same impact, also observed for all other classifiers for state detection. Interesting is, however, the overall smaller body of the boxplots. This indicates less dependence on the chosen filtration distance. In contrast to scenario one, while the choice of filtration distance by mean gives the best median results (median of 0.55), choice by maximum follows closely (median of 0.50) with random choice trailing substantially behind 0.38. This is consistent with our previous deductions regarding the topology of the data. In essence, noise results in short-lived topology features, skewing the random selection. Additionally, there is not a singular defining long-lived feature. Thus, the birth time of the longest-lived feature — while a decent choice — is not the optimal one.

Similar to the first scenario, the better class separability provided by the behavior analysis results in the overall better classification performance of TDABC. With an MCC of 0.94 it achieves similar results to MLP and SVM. The correlation between time steps, extraction method, and MCC also closely follows the trends observed for the other classifiers.

TDA-Autoencoder

As with scenario one, we utilize the autoencoder for binary classification (differentiation between abnormal and benign), employing a fixed percentage of expected abnormal instances to achieve classification. The classifier achieves its best classification (0.61 MCC) employing min-max normalization, 256 neurons for the first, 64 for the second layer, 32 for the third and fourth layer, and 2 neurons for the fifth layer, with a learning rate of 0.05. It performs significantly worse than in scenario one (0.73 MCC) but appears to perform comparable to

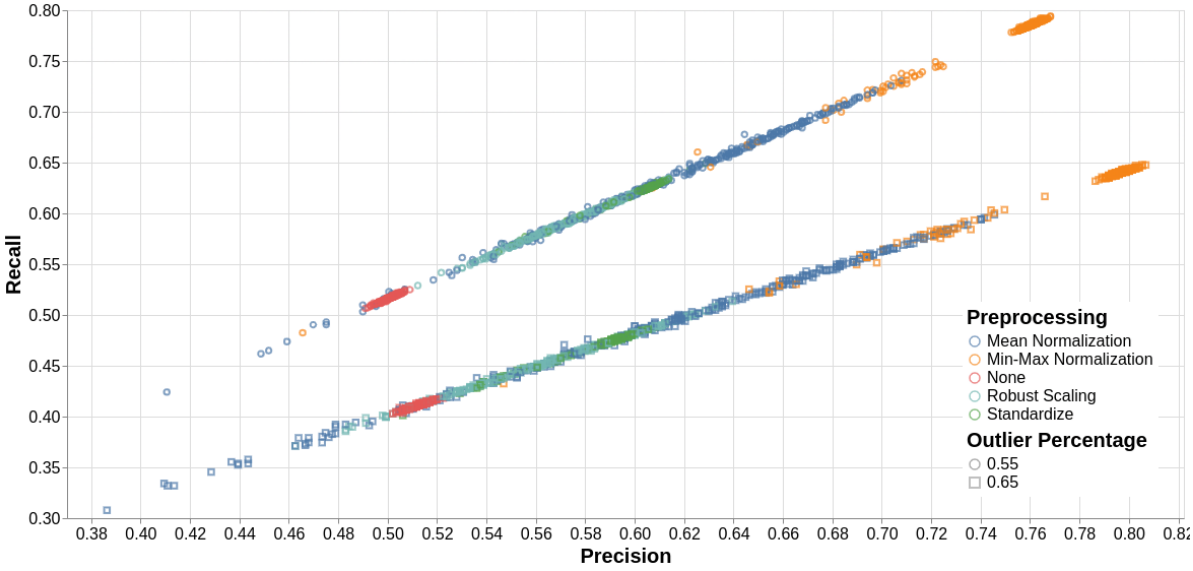


Figure 45: *Precision and recall depending on outlier percentage and preprocessing choice. Each point represents a tested classifier configuration.*

the standard classifiers [2]. In scenario one, we could observe a substantial difference between the classification performance of the autoencoder and SVM, RF, and MLP. The absence thus indicates stronger noise resilience.

A more detailed representation of the performance is given in Figure 45. Note the difference in x and y scales compared to scenario one. We can still observe the negative impact of applying no scaling as caused by the scaling issues of a subset of the features. Interesting is, however, the clear trend between preprocessing methods. While we could not observe any meaningful correlation in scenario one, now min-max normalization decisively outperforms the rest, followed by mean normalization, standardization, and robust scaling.

While this effect is consistent with the greater reliance on preprocessing observed for the standard classifiers (compared to scenario one), the algorithms leading to the best results differ. This is related to the comparatively better performance of this classifier. The regularization employed emphasizes differences, as it is derived from the deviations between the latent and input space topologies. Consequently, just like with the topological feature extraction, this process adds a certain amount of noise resilience. This explains the better comparative performance but intuitively should benefit anomaly preserving methods. However, this ties into what we already observed during the entire evaluation. Namely, that strong anomalous

[2] As the autoencoder performs a binary classification, the respective MCC values can not be directly compared.

values primarily occur in the sensors *network_activity* and *new_process_info*. A number of the other sensors even feature hard limits concerning the possible feature values (e.g., *cpu_load*). As we have seen in section 6.3.2 the sensor importance of especially *new_process_info* has dropped to it being practically irrelevant. Still, these features (especially *network_activity*) and consequently their anomalies hold substantial value, which is why we observe methods like robust scaling and normalizing by mean, performing best for simple state detection. Here, however, we do not perform simple state detection. While not having a strict temporal ordering as in behavior analysis, we still consider multiple states (specifically all within the respective minibatch) simultaneously through the regularization process. Concerning the behavior analysis, we actually can observe a worse performance of anomaly preserving methods (although we can also observe a dependence on the chosen classifier). Accordingly, preprocessing that does not ignore outliers and thus preserves greater detail regarding features without outliers is beneficial.

6.3.5 TDA-based behavior comparison in contrast to traditional approaches

As with scenario one, we compare the quality of the embedding of device behavior achieved with T-ChangeDrift, with the ones generated by the TDA-based extraction methods. For an introduction to T-ChangeDrift and its embedding method, please refer to section 6.2.5. T-ChangeDrift utilizes the structure of the data and consequently, we expect a decent amount of noise resilience. However, we have shown in section 6.2.5 that — in comparison to TDA — T-ChangeDrift's embedding is lossy. Accordingly, we expect a near-identical performance difference as observed in scenario one, with an increased difference for the classes most heavily impacted by the IDS noise. All algorithms are set up as detailed in section 6.2.5.

Assessment

The MCC Δ between our and T-ChangeDrift's approach is shown in Figure 46. In direct comparison to our observations during the first scenario (cf. Figure 37), there are no striking dissimilarities. Neither are there notable differences regarding class performance. Accordingly, our approach proves its capability of extracting higher amounts of information again.

Nonetheless, note the difference between the magnitude of Δ in both scenarios. In particular, the absolute difference in multiclass MCC Δ is 0.01, 0.03, and 0.05 for MLP, SVM, and Random Forest respectively. While the difference is small, we can observe a correlation with expressive power. This implies a slightly lower noise resilience, resulting in worse class separation. In

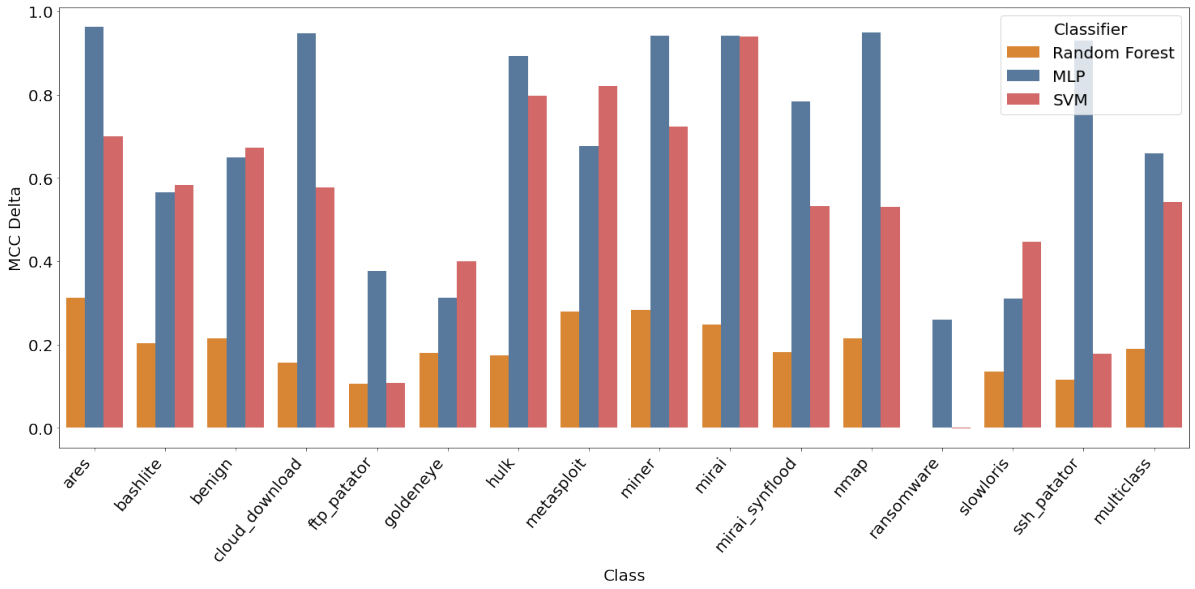


Figure 46: *Difference in MCC per class depending on classifier choice between T-ChangeDrift and TDA extracted behavior (our MCC minus T-ChangeDrift's MCC). In both cases, the best performing classifier configuration is selected.*

particular, the more pronounced differences in classification performance concerning classes most heavily impacted by the noise (e.g., Mirai, Bashlite, Ares), visualize this characteristic. We can consider the class separability index of the unprocessed embeddings to assess this assumption. With a value of 0.95 for TDA feature extraction (specifically defining quantities) versus T-ChangeDrift's 0.81, we can clearly observe the noise's impact.

Overall, these results reaffirm that **H3B** strongly holds, as established in scenario one. While T-ChangeDrift also provides good noise resilience, TDA enables us to extract significantly more relevant information.

6.4 Notable Observations

During both the evaluation of the behavior analysis and the comparison with T-ChangeDrift, we can observe that a time step based analysis can compensate for the loss of information when dropping certain features. Primarily, this concerns the two feature *new_process_info* and *network_activity*. In particular, during the evaluation of T-ChangeDrift, we can observe that dropping these features can even lead to a notable performance improvement. This is although our evaluation of the state-based analysis clearly shows the information gain when introducing

the features in questions. Unfortunately, given the data at our disposal, it is impossible to establish a relationship between the original features and their importance to the classification result after the TDA feature extraction is performed. Thus, we have to conclude regarding **H2**, that the sensors in question only hold value for state-based analysis. Accordingly, this phenomenon requires further research. Given these results, we suggest dropping these features in favor of larger time windows for the behavior analysis.

There is a discrepancy between the expected number of collected data instances and the actual one. This is caused by the high computational overhead of the *new_process_info* in particular. Accordingly, we decided against using other similarly evolved sensors that would increase the overhead further. Notable is the system call sensor already present in the toolset or with regard to process information the capture of the full process tree as we found that any more overhead would be unjustifiable. Regarding the current overhead, we assume a sensor implementation in C — as expected in a real-world scenario — should be enough to compensate for it. Given the nature of the sensor collection, especially the relatively large aggregation intervals, we consider the current amount of information loss not to be detrimental to the significance of the assessment of the usability of topological data analysis in this field of intrusion detection.

Table 11 reports the average runtime for the different classifiers, depending on various experiment setups. Other than the drop in training time of the MLP between scenarios one and two, there are no noteworthy abnormalities. All classifiers share the same trends with regard to the different setups. The behavior of the MLP training can be explained by an increase of the validation interval caused by time constraints coupled with the little to no usage of the fine-grained validation interval in the evaluation of scenario one. Notable is the extremely low training time of the TDABC. This can be explained by the fact that a large part of the calculations needed for classification using TDABC are only performed in the test step and not during training.

Table 11: *Average runtime in seconds depending on various experiment setups.*

Type	Scenario 1 μ	Scenario 1 σ	Scenario 2 μ	Scenario Two σ
State				
MLP	78.3807	1.8058	45.8570	9.7584
RF	0.4997	0.6535	0.4975	0.6555
SVM	1.4350	1.5264	1.8693	1.6145
TDABC	0.0006	0.0005	0.0011	0.0004
TDAENC	1821.9828	704.0092	1872.4608	739.7226
Binning				
MLP	86.6730	2.2892	58.0900	3.2620
RF	0.5163	0.6804	0.4271	0.5653
SVM	0.4983	0.1286	3.0942	0.9068
TDABC	0.0005	0.0006	0.0009	0.0013
Codebook				
MLP	79.9000	3.7812	49.7560	2.6426
RF	0.3729	0.4891	0.4452	0.5747
SVM	1.0036	0.2517	0.7344	0.2513
TDABC	0.0007	0.0005	0.0012	0.0005
Defining Quantities				
MLP	78.0924	2.0210	50.2421	2.6383
RF	0.6389	0.8321	0.5321	0.6991
SVM	0.2467	0.1468	1.4087	0.8055
TDABC	0.0006	0.0005	0.0011	0.0004
Entropy				
MLP	77.2408	1.7855	49.3702	2.6321
RF	0.5038	0.6564	0.3849	0.4892
SVM	0.1461	0.0319	0.7220	0.1403
TDABC	0.0009	0.0006	0.0018	0.0007
Pairwise Distance				
MLP	79.8678	2.0410	52.2118	2.8695
RF	1.2729	1.6630	1.0619	1.3849
SVM	1.2428	0.6277	2.0229	0.9217
TDABC	0.0005	0.0004	0.0009	0.0004

7 Conclusion

This work explores intrusion detection for a set of homogeneous IoT devices. This idea follows ChangeDrift, an abstract concept regarding the detection of divergent behavior within a system of homogeneous nodes. We utilize topological data analysis to facilitate intrusion detection in this context, specifically for sets of homogeneous IoT devices in different scenarios. In particular, we argue that TDA enables the extraction of relevant data with high information retention at various stages of the detection process. Specifically, we consider three distinct stages and evaluate the potential benefits of TDA at each of them. First is the inter-device sensor data fusion (i.e., the extraction of a singular feature vector capturing the magnitude of difference between a set of devices), second the temporal behavior feature extraction (i.e., observation of device behavior over n time steps and encapsulation into singular feature vector) and lastly utilization of TDA concepts directly within the classifiers.

7.1 Tool enabling reproducible generation of IDS Datasets

This evaluation requires the creation of two new datasets to allow full assessment of the merit of the different approaches. Namely, these are a centralized scenario modeled after a typical security camera setup (as found, for example, at an airport), as well as a distributed mobile ad-hoc network (e.g., a vehicular network exchanging sensor information regarding road conditions). To achieve this, we present a new toolset, enabling the reproducible generation of IDS datasets. Our focus is to simultaneously address some issue that we see with many modern and legacy datasets, their reproducibility. Unfortunately, many datasets lack in these regards and thus heavily impacting the ability to adjust the respective datasets. These adjustments could be merited based on, for example, outdated attacks, a specific use case that deviates slightly from the original datasets, or errors in the dataset itself. Hence, the tool is created with a primary focus on reproducibility without impacting flexibility and diversity of the generated datasets.

To this end, the tool comprises multiple modular systems. Primarily, we differentiate between three stages concerning sensor measurements, sensor data distribution, and sensor data collection, respectively. Each stage features further modular systems, enabling the easy addition of new sensors, aggregators, network modules, and the like. To showcase and evaluate the tool with respect to the pre-established goals, as well as to determine its usability for our use case, we utilize the tool to recreate two popular IDS datasets. These are the CICIDS2017 dataset and the ADFA-LD dataset. Former is a popular NIDS dataset featuring a moderately large network setup with a wide variety of different hosts. Latter is a HIDS, collecting system calls of a single device.

The evaluation shows that the recreations are on par with the original datasets, barring insufficiencies that we discovered and subsequently fixed in the ADFA-LD dataset. We assess both recreations by considering their structural properties as well as the performance of classification approaches of reference works on both the original and our reproduction. For CICIDS2017, we are able to closely match the original dataset both in structure as well as classifier performance. It is important to note that we reused the original dataset's benign data instances, as their method to generate these is not publicly available. However, we show that this is unproblematic due to us closely matching the original setup.

The insufficiencies regarding the ADFA-LD dataset primarily concern benign data generation. Here we observe clear signs of stark behavior differences between training and validation data as well as a very narrow set of benign user actions. All this negatively impacts the quality of classifiers trained on the dataset and does not mimic expected benign user behavior. Thus, we address these issues in our reproduction at the cost of comparability. However, note that due to inadequate documentation, with respect to, for example, the difference in training and validation data, comparability is already heavily impacted. While not entirely faithful to the original, the adjusted reproduction still showcases our tool's ability to capture the required data for a HIDS. Thus, we conclude our goals regarding the tool as being met.

7.2 TDA facilitated Intrusion Detection for homogeneous IoT Devices

Employing the tool to generate the two new datasets, we focus on anomaly-based intrusion detection for a set of homogeneous devices. The goal is to utilize their homogeneity to discover devices deviating from expected behavior. Topological data analysis proves to be a very potent tool for this specific use case. We see advantages at all three stages at which we introduce

TDA into the detection process (cf. chapter 6). In particular, the first two stages prove to be highly successful. With respect to the third stage, while we can observe promising results, the characteristics of the given datasets benefit the more traditional classifiers evaluated.

The first stage employs TDA to achieve a comparison between complex sensor values that are representable as a point cloud. We utilize this to use sensors closely monitoring new processes as well as network activity, next to standard, single value sensors such as CPU utilization. The TDA approach proves to be able to retain sensor information excellently; however, we can observe a high impact of noise on the sensors themselves.

We present the benefit of TDA-based feature extraction to achieve device behavior analysis over a larger timespan in stage two to address the aforementioned issue. Here, we can observe not only great performance on the relatively noise-free centralized dataset but also outstanding noise resilience with as little as 15 time steps in the distributed case (depending on the feature extraction method). We especially can observe an excellent class separation. Note that due to this, the higher computational cost of the feature extraction can be offset by employing a simpler classifier.

For both stages one and two, we can observe the benefits of employing TDA regarding detecting largely covert acting attacks. Thus, we hypothesize potential advantages of TDA-based classifiers. However, while we can observe greater noise resilience, the tested algorithms fall short of its traditional counterparts. Thus, we deem them unfit for this specific use case.

As our approach falls into the ChangeDrift ecosystem, we evaluate our TDA-based extraction approach against a comparable work, also performing anomaly-based intrusion detection in homogeneous IoT networks but employing hierarchical agglomerative clustering instead (T-ChangeDrift). While hierarchical agglomerative clustering shares many similarities with persistent homology concerning zero-dimensional topological features, we prove that information provided by higher-dimensional features has high relevance. Classifiers trained on our sensor data embedding clearly outperform T-ChangeDrift's embedding regardless of attack class. In particular, we show that we are able to achieve both better class separability and information retention than T-ChangeDrift. Accordingly, TDA-based feature extraction proves to be beneficial both in the supervised scenarios evaluated here but also in an unsupervised one, as presented in T-ChangeDrift.

7.3 Future Work

As noted throughout this thesis, a few aspects require future work. These concern three aspects of this work. During the evaluation, we ran into runtime problems. Especially, concerning the *new_process_info* sensor. Here possible future work could optimize the sensor and, in particular, consider ways to extend it. Furthermore, we can observe little additional value being provided by the sensor in comparison to the *network_activity* one in its current configuration. However, we still believe process information carries significant information. But, due to these computational limitations, we were not able to consider more comprehensive versions of this sensor (e.g., capturing the entire process tree). Additionally, the tool can be further extended regarding both sensor and aggregator modules.

The second aspect allowing for potential future work is the standard classifiers, particularly concerning anomalous behavior detection. Here we observe that the TDA feature extraction is capable of providing excellent class separation. As we noted during the evaluation, this can be utilized to offset the computational cost of the extraction. To this end, the performance of simpler classifiers have to be evaluated. Additionally, based on the good information retention, we hypothesize benefits of this approach regarding unsupervised anomaly detection, as performed by T-ChangeDrift.

Lastly, the evaluation of the topological autoencoder is — due to time constraints — not fully completed. In particular, its performance for behavior analysis as well as its embedding capabilities (i.e., utilizing its latent representation for further classification) has yet to be evaluated.

References

[AK15] Abomhara, Mohamed & Køien, Geir M.: "Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks". In: *J. Cyber Secur. Mobil.* 4, 2015, pp. 65–88.

[Ang17] Angrishi, Kishore: "Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets". In: *ArXiv*, 2017.

[Bil05] Bille, Philip: "A survey on tree edit distance and related problems". In: *Theoretical Computer Science* 337.1, 2005, pp. 217–239. URL: https://www.sciencedirect.com/science/article/pii/S0304397505000174.

[Bio22] Biondi, Philippe: *scapy*. https://scapy.net/. Accessed: 14 February 2022. 2022.

[Bla21] Blaine, Geoff: *Tipping Point: SonicWall Exposes Soaring Threat Levels, Historic Power Shifts In New Report*. https://blog.sonicwall.com/en-us/2021/03/sonicwall-exposes-soaring-threats-historic-power-shifts-in-new-report/. Accessed: 14 February 2022. 2021.

[Boe18] Boes, Felix. unpublished. 2018.

[BP19] Boissonnat, Jean-Daniel & Pritam, Siddharth: "Computing Persistent Homology of Flag Complexes via Strong Collapses". In: *35th International Symposium on Computational Geometry (SoCG 2019)*. Ed. by Gill Barequet & Yusu Wang. Vol. 129. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 55:1–55:15. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10459.

[Bra18] Brailsford, Maxim ZavodchikLiron SegalAaron: *New Python-Based Crypto-Miner Botnet Flying Under the Radar*. https://www.f5.com/labs/articles/threat-intelligence/new-python-based-crypto-miner-botnet-flying-under-the-radar. Accessed: 14 February 2022. 2018.

[Bun18] Bungartz, Christian: "Anomalieerkennung mit Hilfe neuronaler Netze". In: 2018.

[BW14] Blowers, Misty & Williams, Jonathan: "Machine Learning Applied to Cyber Operations". In: vol. 55. 2014, pp. 155–175.

[Can+15] Cang, Zixuan; Mu, Lin; Wu, Kedi; Opron, Kristopher; Xia, Kelin & Wei, Guo-Wei: "A topological approach for protein classification". In: *Molecular Based Mathematical Biology* 3, 2015.

[Car21] Carter, N. (Ed.): *Data Science for Mathematicians*. 1st ed. Chapman and Hall/CRC, 2021.

[Cat+21] Catillo, Marta; Del Vecchio, Andrea; Ocone, Luciano; Pecchia, Antonio & Villano, Umberto: "USB-IDS-1: a Public Multilayer Dataset of Labeled Network Flows for IDS Evaluation". In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 2021, pp. 1–6.

[CH13] Creech, Gideon & Hu, Jiankun: "Generation of a new IDS test dataset: Time to retire the KDD collection". In: *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. 2013, pp. 4487–4492.

[Chi+15] Chintakunta, Harish; Gentimis, Thanos; Gonzalez-Díaz, Rocio; Jiménez, María & Krim, Hamid: "An entropy-based persistence barcode". In: *Pattern Recognition* 48, 2015.

[CW17] Cang, Zixuan & Wei, Guo: "Integration of element specific persistent homology and machine learning for protein-ligand binding affinity prediction". In: *International Journal for Numerical Methods in Biomedical Engineering* 34, 2017.

[Dia+17] Dias, L. P.; Cerqueira, J. J. F.; Assis, K. D. R. & Almeida, R. C.: "Using artificial neural network in intrusion detection systems to computer networks". In: *2017 9th Computer Science and Electronic Engineering (CEEC)*. 2017, pp. 145–150.

[Hab+17] Habibi Lashkari, Arash; Draper Gil, Gerard; Mamun, Mohammad & Ghorbani, Ali: "Characterization of Tor Traffic using Time based Features". In: 2017, pp. 253–262.

[Haw+02] Hawkins, Simon; He, Hongxing; Williams, Graham & Baxter, Rohan: "Outlier Detection Using Replicator Neural Networks". In: *Data Warehousing and Knowledge Discovery*. Ed. by Yahiko Kambayashi; Werner Winiwarter & Masatoshi Arikawa. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 170–180.

[Ho95] Ho, Tin Kam: "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Vol. 1. 1995, 278–282 vol.1.

[Hof+17] Hofer, Christoph; Kwitt, Roland; Niethammer, Marc & Uhl, Andreas: "Deep Learning with Topological Signatures". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1633–1643.

[ice21] iceland2k14: *solominer*. https://github.com/iceland2k14/solominer/. Accessed: 14 February 2022. 2021.

[Kha+19] Khan, Khalid; Mehmood, Amjad; Khan, Shafiullah; Altaf, Muhammad; Iqbal, Zeeshan & Mashwani, Wali: "A survey on Intrusion Detection and Prevention in Wireless Ad-hoc Networks". In: *Journal of Systems Architecture* 105, 2019, p. 101701.

[Kin+21] Kindelan, Rolando; Frias, Jose; Cerda, Mauricio & Hitschfeld, Nancy: "Classification based on Topological Data Analysis". In: *CoRR*, 2021. arXiv: 2102.03709. url: https://arxiv.org/abs/2102.03709.

[Kol+17] Kolias, Constantinos; Kambourakis, Georgios; Stavrou, Angelos & Voas, Jeffrey: "DDoS in the IoT: Mirai and Other Botnets". In: *Computer* 50.7, 2017, pp. 80–84.

[Kre17] Krebs, Brian: *Who is Anna-Senpai, the Mirai Worm Author?* https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/. Accessed: 14 February 2022. 2017.

[Luq+19] Luque, Amalia; Carrasco, Alejandro; Martín, Alejandro & de las Heras, Ana: "The impact of class imbalance in classification performance metrics based on the binary confusion matrix". In: *Pattern Recognition* 91, 2019, pp. 216–231. url: https://www.sciencedirect.com/science/article/pii/S0031320319300950.

[Ma+18] Ma, Shiqing et al.: "Kernel-Supported Cost-Effective Audit Logging for Causality Tracking". In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, 2018, pp. 241–254. url: https://www.usenix.org/conference/atc18/presentation/ma-shiqing.

[Man22] Mangalapilly, Yesudeep: *watchdog*. https://pythonhosted.org/watchdog/. Accessed: 14 February 2022. 2022.

[MM08] Mthembu, Linda & Marwala, T.: "A note on the separability index". In: *arXiv: Methodology*, 2008.

[Moo+20] Moor, Michael; Horn, Max; Rieck, Bastian & Borgwardt, Karsten: "Topological Autoencoders". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III & Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 7045–7054. URL: https://proceedings.mlr.press/v119/moor20a.html.

[MU13] Mohamad, Ismail & Usman, Dauda: "Standardization and Its Effects on K-Means Clustering Algorithm". In: *Research Journal of Applied Sciences, Engineering and Technology* 6, 2013, pp. 3299–3303.

[MY10] Mustaffa, Zuriani & Yusof, Yuhanis: "A comparison of normalization techniques in predicting dengue outbreak". In: vol. 1. 2010, pp. 345–349.

[NAA20] Nguyen, Minh; Aktas, Mehmet & Akbas, Esra: "Bot Detection on Social Networks Using Persistent Homology". In: *Mathematical and Computational Applications* 25.3, 2020. URL: https://www.mdpi.com/2297-8747/25/3/58.

[Osm+20] Osman, Amr; Wasicek, Armin; Köpsell, Stefan & Strufe, Thorsten: "Transparent Microsegmentation in Smart Home IoT Networks". In: *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, 2020. URL: https://www.usenix.org/conference/hotedge20/presentation/osman.

[Ott+15] Otter, Nina; Porter, Mason; Tillmann, Ulrike; Grindrod, Peter & Harrington, Heather: "A roadmap for the computation of persistent homology". In: *EPJ Data Science* 6, 2015.

[Ped+11] Pedregosa, F. et al.: "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12, 2011, pp. 2825–2830. URL: http://dl.acm.org/citation.cfm?id=1953048.2078195.

[Rod22] Rodola, Giampaolo: *psutil*. https://psutil.readthedocs.io/en/latest/. Accessed: 14 February 2022. 2022.

[Sea21] Seals, Tara: *IoT Attacks Skyrocket, Doubling in 6 Months*. 2021.

[SF09] Shameem, Mushfeq-Us-Saleheen & Ferdous, Raihana: "An efficient k-means algorithm integrated with Jaccard distance measure for document clustering". In: *2009 First Asian Himalayas International Conference on Internet*. 2009, pp. 1–6.

[Sha+17] Sharafaldin, Iman; Gharib, Amirhossein; Habibi Lashkari, Arash & Ghorbani, Ali: "Towards a Reliable Intrusion Detection Benchmark Dataset". In: *Software Networking* 2017, 2017, pp. 177–200.

[SHG18] Sharafaldin, Iman; Habibi Lashkari, Arash & Ghorbani, Ali: "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: 2018, pp. 108–116.

[SMC07] Singh, Gurjeet; Memoli, Facundo & Carlsson, Gunnar: "Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition". In: *Eurographics Symposium on Point-Based Graphics*. Ed. by M. Botsch; R. Pajarola; B. Chen & M. Zwicker. The Eurographics Association, 2007.

[SRD20] Sengupta, Jayasree; Ruj, Sushmita & Das Bit, Sipra: "A Comprehensive Survey on Attacks, Security Issues and Blockchain Solutions for IoT and IIoT". In: *Journal of Network and Computer Applications* 149, 2020, p. 102481.

[Ten21] Tennié, Jonas: *ChangeDrift: Anomaly Detection based on Divergent Behaviour of Homogeneous IoT Devices*. 2021.

[Wan+17] Wang, Aohui; Liang, Ruigang; Liu, Xiaokang; Zhang, Yingjun; Chen, Kai & Li, Jin: "An Inside Look at IoT Malware". In: *Industrial IoT Technologies and Applications*. Ed. by Fulong Chen & Yonglong Luo. Cham: Springer International Publishing, 2017, pp. 176–186.

[WCC04] Wang, Junping; Chen, Quanshi & Chen, Yong: "RBF Kernel Based Support Vector Machine with Universal Approximation and Its Application". In: *Advances in Neural Networks – ISNN 2004*. Ed. by Fu-Liang Yin; Jun Wang & Chengan Guo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 512–517.

[XH13] Xie, Miao & Hu, Jiankun: "Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD". In: *2013 6th International Congress on Image and Signal Processing (CISP)*. Vol. 03. 2013, pp. 1711–1716.

[Yin+17] Yin, C.; Zhu, Y.; Fei, J. & He, X.: "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks". In: *IEEE Access* 5, 2017, pp. 21954–21961.

[ZJZ21] Zieliński, Bartosz; Juda, M. & Zeppelzauer, M.: "Persistence Codebooks for Topological Data Analysis". In: *Artif. Intell. Rev.* 54, 2021, pp. 1969–2009.

List of Figures

List of Tables

Listings