POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Microservices-Based Autonomous Anomaly Detection for Mobile Network Observability

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Tommaso Brumani**

Student ID: 994233
Advisor: Prof. Davide Martinenghi
Co-advisors:
Academic Year: 2022-23

Abstract

In modern telecommunication networks, network observability entails the use of diverse data sources to understand the state and behavior of the network, and its ability to provide the required service and user experience. Because of the vast amounts of data collection and transmission involved in this process, the network's performance is negatively impacted, and it can become difficult for network operators to identify the occurrence of problematic behavior before it is too late.

To enable a more efficient form of data collection and aid in diagnostic operations, this thesis aims to develop an autonomous anomaly detection system for time series data. The system is to be developed as a microservices-based solution, to be integrated with a software-defined networking controller platform developed at *Ericsson*.

This thesis describes the extensive experimentation process conducted during the development of this system, including various methods of data processing, time series clustering, and anomaly detection. The resulting system is a highly customizable and scalable product, supported by modern and reliable anomaly detection models. The system is capable of detecting several different kinds of anomalies in an arbitrary n umber of mobile network monitoring metrics, and can be easily configured to fit the specific needs of each customer.

Keywords: anomaly detection, outlier detection, time series, software-defined networking, microservices

Abstract in lingua italiana

Nelle reti di telecomunicazioni moderne, l'osservbilità della rete prevede l'utilizzo di varie fonti di dati per comprendere lo stato e comportamento della rete, e la sua capacità di fornire il servizio e l'esperienza dell'utente richieste. A causa delle vaste quantità di dati che è necessario raccogliere e trasmettere come parte di questo processo, le prestazioni della rete risultano deteriorate, e puó essere difficile per gli operatori della rete identificare occorrenze di comportamenti problematici prima che sia troppo tardi.

Per permettere una forma piú efficiente di raccoglimento dei dati e per aiutare nelle operazioni diagnostiche, questa tesi si pone l'obbiettivo di sviluppare un sistema di rilevamento delle anomalie autonomo per dati aventi la forma di serie temporali. Is sistema é implementato mediante un'architettura a microservizi, da essere integrato con una piattaforma software-defined networking controller sviluppata ad *Ericsson*.

Questa tesi descrive l'ampio processo di sperimentazione condotto durante lo sviluppo del sistema, che include varie operazioni di trattamento dei dati, clustering di serie temporali, e rilevamento di anomalie. Il sistema risultante é un prodotto altamente scalabile e personalizzabile, supportato da sistemi di rilevamento delle anomalie moderni e affidabili. Questo sistema é in grado di rilevare diversi tipi di anomalie in un numero arbitrario di metriche per il monitoraggio della rete, e puó essere facilmente configurato per adattarsi ai bisogni specifici di ciascun cliente.

Parole chiave: rilevamento delle anomalie, rilevamento di valori anomali, serie temporali, software-defined networking, microservizi

Contents

1 | Introduction

The following chapter provides an introductory view of the thesis. Section 1.1 describes the motivation behind this work, as well as an outline of how its implementation was conducted. Section 1.2, instead, provides an overview of the content of the chapters that are to follow.

1.1. Problem Statement

In modern telecommunication networks, network observability entails the use of diverse data sources to understand the state and behavior of the network, and its ability to provide the required service and user experience.

Over the course of recent years, mobile transport networks have become increasingly complex and difficult to manage, leading to the development of new approaches to centralize the management of the network, such as software-defined networking. As a result, a common problem related to network observability is the large quantity of data that is required to comprehensively analyze and observe the network behavior, and the load the collection and transmission of that data puts on the network's resources.

Collecting such large amounts of detailed data, however, is superfluous during the majority of the network's operations. A detailed description of the network's state is of most interest only when the network behaves unusually, so that they may be used to understand the root cause of the disturbance. Because of this, most data should only be collected when anomalies arise in the network, but the extensive size of these networks and the frequent changes to which they are subjected make it unfeasible for this collection to be triggered by human agents. Furthermore, the vast amount of collected data makes it difficult for network operators to recognize when and which parts of the network may be affected by anomalous events, making analysis possible only after the event has visibly impacted the network's operations.

To solve these problems, a small subset of diagnostic network metrics influenced by anomalies of interest can be selected, and an anomaly detection system can be designed to au-

tonomously detect disturbances in their behavior. Such a system would make it possible to automatically request more detailed data from the affected network elements when unusual behavior takes place, and immediately alert network operators whenever the system starts to diverge from its established patterns. What makes this task particularly difficult to accomplish, however, is the high degree of variation that can be observed in the values taken over time by network metrics. These metrics are often affected by various kinds of cyclical variations, as well as permanent alterations to their behavior in response to changes in the environment and network configuration, which make most basic detection techniques insufficient. As a result, an anomaly detection system operating on such data should be capable of detecting anomalies in metrics characterized by very different behavioral patterns, be adaptable to changes in their behavior, and be able to function in the presence of previously unseen patterns.

1.1.1. Research Question

The purpose of this thesis is to answer the following research question:

RESEARCH QUESTION

How could an anomaly detection system for reduced amounts of mobile network operational data be constructed?

More specifically, the goal is to design an autonomous system for detecting the presence of anomalies in some basic network time series data, in order to notify operators, enable the automatic adjustment of data collection, and other forms of automation. The implementation should be a cloud-native, microservices-based solution to be integrated on top of an existing software-defined networking controller platform developed at *Ericsson*.

The use of microservices ensures that the system may be easily integrated with the platform while undergoing development independently, so that it may be maintained and updated without impacting the main system. The detection system should be built using reliable and scalable technologies, and constructed to be easily configurable to suit the needs of the end user, as well as to be easy to expand through future work.

1.2. Structure of the Thesis

The rest of this thesis is organized as follows:

- **Chapter 2** explains in greater detail the fundamental concepts of networking and

anomaly detection necessary for the understanding of the problem at hand, as well as its relevance. It also details the context in which this work was conducted, including the software utilized and the characteristics of the available data set and hardware, and provides a high-level overview of the solution to be developed.

- **Chapter 3** contains a theoretical explanation of the variety of methods applied during the implementation phase, including data processing, clustering, and machine learning techniques. It also describes the choice of strategy used to evaluate the various tested algorithms, and the metrics employed for this purpose.
- **Chapter 4** details the implementation of the thesis itself, such as the processing operations conducted on the available data set, various attempts at clustering the processed data, and experimentation with various machine learning libraries. It also contains a numerical evaluation of the various tested models, and a description of the final implemented system.
- **Chapter 5**, finally, summarizes the contents of this document, discusses its success in achieving its stated goals, and lists a number of future developments that may be carried out to further improve on its results.

2 | Problem Setting

The following chapter expands on the problem statement expressed in Chapter 1, and provides the foundations necessary for the understanding of the methods described in Chapter 3. It explains the most relevant domain concepts, as well as the specific context in which this thesis was carried out. Section 2.1 describes the approach of software-defined networking, as well as its impact on the internet protocol networking industry. Section 2.2 describes the principles of microservices architecture, and the reasons that make it well-suited for the implementation of the anomaly detection systems. Section 2.3 describes the concept of network observability, including why it is relevant and why it is difficult to achieve. Section 2.4, describes the fundamental concepts of anomaly detection and time series. Section 2.5 describes the data set used as part of this study, and the rationale behind the decisions made in its regard. Section 2.6 describes the computational resources and software libraries used during the experimental phases of this thesis work. Section 2.7, finally, provides a high-level overview of the system to be developed, and lists the principles followed in its design and implementation.

2.1. Software Defined Networking

Despite the widespread adoption of the Internet, traditional Internet protocol (IP) networks are complex and difficult to manage. [4] This is derived from a number of issues, including the difficulty in configuring the network according to predefined policies as well as in response to faults or changes. This is because expressing the desired high-level network policies requires each individual network device (routers, switches, etc.) to be configured separately using low-level, vendor-specific commands. [33] Another source of complexity is the high degree of vertical integration often encountered in these networks: the control plane (which decides how to handle the traffic) and the data plane (which forwards traffic according to the control plane's decisions) are tightly joined, reducing flexibility and stifling innovation. [44]

Software-defined networking (SDN), is an approach to design, implement, and manage IP networks that entails the separation of the control plane and the data plane, thereby allow-

ing programmatically efficient network configuration by centralizing network intelligence in the network controllers. [5] In this way, network nodes become simple traffic-forwarding devices that communicate with the SDN controller by means of established application programming interfaces (API), which simplifies policy enforcement and network configuration and evolution. [30] A graphical representation of SDN architecture is provided in Figure 2.1.

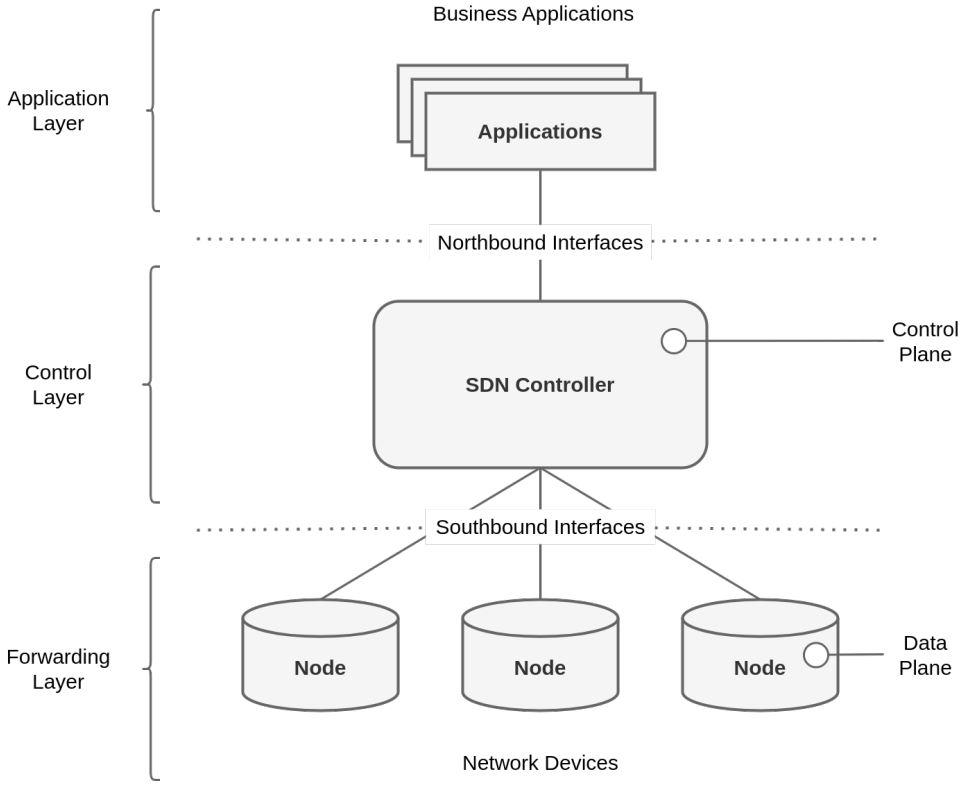


Figure 2.1: Simplified graphical representation of SDN architecture.

According to Kreutz et al. [33], the pillars of SDN network architecture are that:

1) The control plane and data planes are decoupled.

2) Traffic forwarding decisions are flow-based (not destination-based), where a flow is intended as a sequence of packets between a source and a destination that are handled with identical service policies on the forwarding devices.

3) Control logic is moved to an external entity, the SDN controller, which provides an abstracted network view that simplifies the programming of forwarding devices.

4) The network is programmable by way of software applications running on the SDN controller.

The integration of the controller with software applications is one of the fundamental elements of SDN's value proposition, and makes it possible to execute complex operations over the network infrastructure, such as deploying an anomaly detection system (ADS). In particular, SDN is suitable for the use of more advanced anomaly detection techniques based on machine learning because of its ability to obtain a global view of the network. [58] The programmability of SDN also ensures that these solutions can be executed on the IP network in real-time. [59]

2.2. Microservices Architecture

Microservices architecture (MSA) is a technique for developing software that makes it possible to structure applications as collections of small, loosely coupled software services. In MSA, these services should be independent processes interacting via messages, deployed in isolation, and possessing dedicated data storage tools. [20] MSA does not mandate any specific programming paradigm, but instead provides a framework to divide the components of a distributed application into independent entities, each one carrying out a specific function necessary to the application's purpose. As a result, the internal implementation of each microservice does not have to abide by any explicit rule, so long as it provides its functionalities to an outward-facing API.

The adoption of MSA is of particular benefit for the development of SDN controllers, as it provides the ability to easily scale the controller's resources to match the requirements dictated by the network's scale. Furthermore, it allows for increased flexibility when developing applications such as ADSs, as they are decoupled from a specific controller and can be integrated to expand the latter's functionalities without the need to recompile the controller itself. [18]

2.3. Network Observability

Network observability can be defined as the ability to answer any question about the network quickly and easily. Network monitoring generally consists of two steps: data measurement and data processing. The former includes collecting and storing the data, and can be further divided into collection, preprocessing, and transmission. The latter focuses on organizing the data as interpretable information, and can be split into analysis and presentation. [52] A more detailed description of this process is presented as follows:

1) **Collection**: the process of gathering data is defined by the means through which the data is collected, the target devices to be observed, and the frequency with

which the information is updated.

2) **Preprocessing**: the raw data is aggregated and processed into some statistical format, to aid in itemizing and tracking the measurements. A number of operations may also be performed on the data to make it more suitable for the analysis process that is to follow.

3) **Transmission**: the itemized data is transmitted to the analytics center according to some messaging protocol.

4) **Analysis**: the data is analyzed to produce statistics and identify particular events. The result of this analysis provides network status information to traffic engineering and fault management applications.

5) **Presentation**: the results of the analysis process are presented in a human-readable way, commonly in the form of graphs and tables.

As IP networks increase in size and complexity, however, monitoring increases in difficulty: the latency and inconsistency of large networks can lead to faulty measurements [54], and the difficulty of coordinating measurement collection from various devices mandates efficient methods for preprocessing and transmission.

The advent of SDN architectures has made network monitoring through the collection of measurements from different network devices trivial, providing network agents with a global view that allows them to perform monitoring at different spatial and temporal scales. [57] Transmission of the data is handled through the controller's southbound API, allowing network designers to define suitable data structures for their requirements instead of having to rely on standard communication protocols, and analysis and presentation are made increasingly convenient by allowing for centralized control and integration with applications through the northbound APIs. [52] Such applications include ones for anomaly detection, which may provide additional insights into the outlier events occurring in the network by drawing attention to and helping categorize them.

2.4. Basics of Anomaly Detection

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior, which are often referred to as anomalies or outliers. According to Hawkins [25], an outlier can be defined as: "An observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism". Outlier detection finds use in a large variety of fields due to the assumption that, in a

wide number of application domains, anomalies in data often correspond to relevant and actionable information. [13] The output of an outlier detection algorithm can generally be one of two types [1]:

- **Outlier scores**: a score quantifying how much each data point diverges from the expected behavior.
- **Binary tables**: a label indicating whether a data point is an outlier or not, typically achieved by setting a threshold on outlier scores.

It is, however, worth mentioning the limitations inherent to all anomaly detection techniques: while these systems may be able to identify probable anomalies, they will never be able to ascertain their presence with absolute certainty. Furthermore, they may only provide judgment on the metrics used to monitor the system, but this may or may not correspond to the presence of an anomaly or fault in the underlying system itself. [29]

Keeping this in mind, anomaly detection systems may nevertheless provide great insights into the behavior and health of monitored systems, bring to light undetected problems, help in diagnosing faults and changes, and aid in root cause analysis.

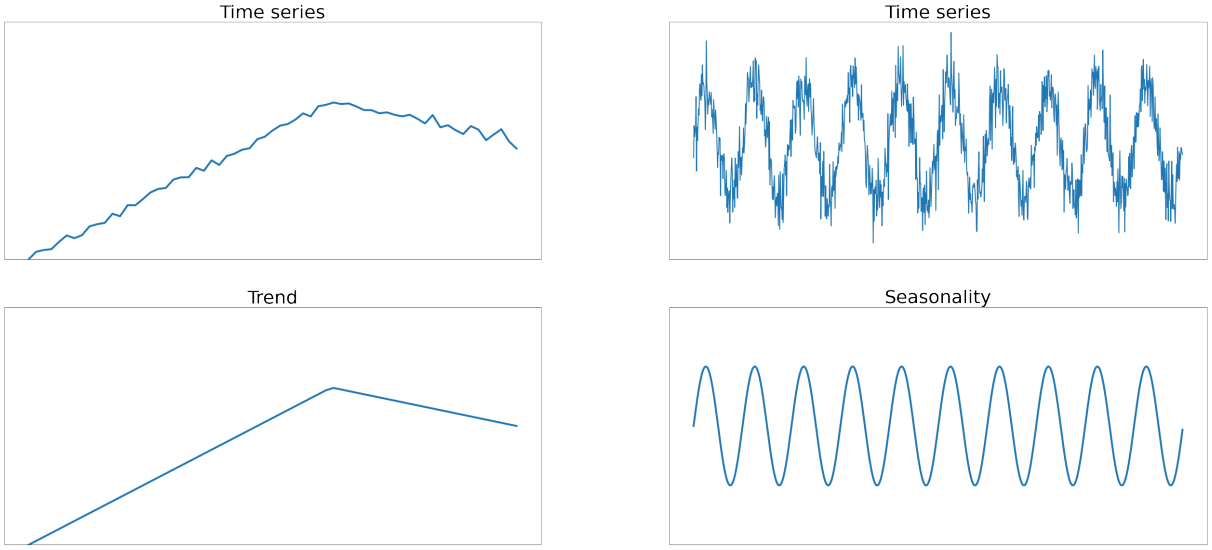
2.4.1. Time Series

One of the many domains of application for anomaly detection is that of time series, for example for the purpose of system monitoring and control. A time series can be defined as a sequence of observations performed in succession through time, either continuously or at discrete instants. For discrete time series, data is typically collected at regular time intervals, and is either sampled from another continuous series, aggregated over a period of time, or is an inherently discrete sequence. [15]

Time series data can be stationary or non-stationary. A stationary process can be defined as a stochastic process for which the unconditional joint probability distribution is not variable with time. [22] A common reason for non-stationarity is the presence of a trend, a low-frequency variation in the mean that may or may not be deterministic. When the trend is deterministic, the series can be transformed into a stationary process by removing the trend component, which is in this case only a function of time, while non-deterministic trends can instead be removed through differencing.

Another possible characteristic of time series is seasonality, the occurrence of repetitive patterns in the series' levels at regular intervals, usually quarterly, monthly, or weekly. A similar concept to that of seasonality is that of cyclical patterns. These patterns differ from seasonal cycles in that they do not repeat with fixed periodicity. Examples of time

series with trend and seasonality are displayed in Figure 2.2.



(a) Time series with a trend.

(b) Time series with seasonality.

Figure 2.2: Examples of time series with trend and seasonality. In both cases, the plot above displays the time series itself, while the one below displays its trend or seasonality in isolation. The x-axis represents time, while the y-axis represents the value of the time series.

According to Jinka et al., [29], the increasing amount of time series data being collected in applications such as IP networks means that traditional means of monitoring are no longer adequate. The easiest way to identify anomalies, for example, used to be the visual inspection of some graphical representation of the data, which requires human intervention and cannot scale up to systems as large as those managed by an SDN controller. Another straightforward method of anomaly detection is the use of static thresholds, but this approach is also severely lacking, as the normal parameters of one system may be different from those of others, and they might also change over time within the same system both on the long and short terms. The shortcomings of these approaches highlight the need for more advanced techniques, capable of acting autonomously, scaling up efficiently, and adapting to the nature of the specific time series that they are applied to.

2.4.2. Classification of Anomaly Detection Approaches

A large body of work has been published classifying, reviewing, and comparing various outlier detection techniques for time series data. [14] [1] [19] [9] [7] In his book on outlier analysis, Aggarwal [1] categorizes anomaly detection methods into probabilistic and statistical models, linear models, proximity-based models, high-dimensional models, and

ensemble models.

Braei and Wagner [9], on the other hand, focus on univariate time series and favor a broader distinction, dividing the methods into statistical methods, classical machine learning methods, and deep learning methods (identified as those machine learning methods that rely on neural networks). The first group of models generally assumes that the data is generated by a specific statistical model, while the second and third treat the data generation process as a black box and attempt to model the data by learning from the observations directly, either through conventional machine learning or deep learning.

In the time series domain, Blazquez et al. [7] instead classify outlier detection techniques based on three factors: their input, outlier type, and nature of the method.

- The input factor specifies the type of data that the detection algorithm can work with. It may be either univariate or multivariate, depending on whether the input is merely an ordered set of observations, each recorded at a given time, or a set of n-dimensional vectors each made up of n observations.
- The method factor specifies the nature of the detection process employed by the algorithm. It can once again be univariate if it considers a single time-dependent variable at a time, or multivariate if it is capable of analyzing multiple variables at once.
- The outlier type factor specifies the kind of anomaly that the algorithm aims to detect: point outliers, subsequence outliers, or outlier time series.

In this classification, point outliers are defined as data points characterized by unusual behavior at a specific time instant when compared to other values in the time series. Subsequence outliers are instead described as sets of consecutive points whose collective behavior is unusual, even though the single point's behavior might not be anomalous on its own. Finally, outlier time series are identified as entire variables whose behavior diverges from the rest. The first two kinds of anomalies may be either univariate or multivariate depending on the number of affected variables, whereas the latter may only occur in multivariate data.

Cook et al. [19] provide a similar classification of outliers into point anomalies, contextual anomalies, and pattern anomalies, based on their relation with surrounding data. Point anomalies are characterized by the return of the time series to its normal behavior within a few observations from the occurrence of the anomalous event. Contextual anomalies are observations that diverge from the regular patterns of the sequence when taken in the context of surrounding data, but fall within the range of expected values when taken in

isolation. Pattern anomalies, finally, are sets of observations that are anomalous when taken as a whole and compared to the rest of the data, while individually they may or may not be unusual.

Another important distinction for anomaly detection methods is that between supervised, semi-supervised, and unsupervised approaches. [9] In supervised approaches the time series data set is labeled, meaning that for each sample it is known whether that sample is an anomaly or not. Semi-supervised approaches are instead characterized by a data set that contains exclusively data points with normal behavior, and no outliers. Unsupervised methods, finally, assume that no label is provided for the data. For this latter type of technique, a common approach is to assign an anomaly score to each data point, and classify as outliers those points whose score's distance from the global mean of anomaly scores exceeds 3 standard deviations.

2.5. Data Set

The raw data set utilized to conduct this study consists of measurements sampled at regular, 15-minute intervals (96 samples per day), from a real-world IP network belonging to one of *Ericsson*'s customers. The samples in the data were collected over the course of several months, from July 2022 to December 2022, and were deemed by company experts to be representative of the type of data the ADS would be operating on after deployment.

Each row of the data describes an observation taken from a single interface of one network element, such as a *router* or *microwave transmitter antenna*. Each of these samples includes several fields, including information regarding the context in which the observation was performed and a `counters` column, a set of measurements of various metrics, calculated over the previous 15-minute interval. For example, a row of the data set might contain the counter `inoctets`, which describes the number of bytes received by the interface during the previous 15 minutes. The number and type of distinct counters present in each row of data varies greatly between different interfaces, ranging from zero to 170 distinct counters for each observation.

The schema of the raw data set is reported in more detail in Figure 2.3, and an example of ten random rows of data is provided in Figure 2.4. Furthermore, an explanation of the contents of the various fields in the schema is provided in Table 2.1.

The data set presents numerous instances of missing measurements in which an interface's records at a given timestamp are completely missing, in some cases resulting in several days of consecutively missing samples for the same interface. More importantly, the data

```
root
 |-- fdn: string (nullable = true)
 |-- neid: string (nullable = true)
 |-- moClass: string (nullable = true)
 |-- granularity: integer (nullable = true)
 |-- software: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- counters: map (nullable = true)
 |    |-- key: string
 |    |-- value: long (valueContainsNull = true)
 |-- pdfCounters: map (nullable = true)
 |    |-- key: string
 |    |-- value: array (valueContainsNull = true)
 |    |    |-- element: long (containsNull = true)
 |-- runid: string (nullable = true)
```

Figure 2.3: Schema of the raw data set, as provided by `PySpark`.

fdn	neid	moClass	granularity	software	timestamp	counters	pdfCounters	runid
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	TermPointToSGW	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{}	{pmS1uPktRecOooQc...	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	ExternalENodeBFun...	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmEenbPktDiscDlC...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	GUtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoPrepAtt -> 0...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	EUtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoOscInterF ->...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	UtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoPrepAttLb ->...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	EUtranFreqRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmAtoMeasuredUe ...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	EUtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoOscInterF ->...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	UtranFreqRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmLbMeasuredUe -...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	UtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoPrepAttLb ->...	{}	datastream
SubNetwork=LTE,Me...	SubNetwork=LTE,Me...	EUtranCellRelation	900	CXP2010174_1 R50K10	2023-01-30 15:30:00	{pmHoOscInterF ->...	{}	datastream

Figure 2.4: Ten rows of the raw data set, chosen randomly.

set contains no information regarding which data points may or may not correspond to anomalies in the underlying system, meaning that the data set can be considered unlabeled for the purpose of anomaly detection. These characteristics were determined to be typical of data collected from mobile transport networks, since these systems can often be subject to equipment malfunction that causes missing measurements, and labeling the data is extremely difficult as a consequence of the vast quantity collected at each sampling instant.

Furthermore, the relatively large amount of time elapsing between consecutive measurements means that finer changes in the counter metrics' behavior cannot be detected. Nevertheless, because the ADS is tasked with operating on a small amount of data and the collection of finer-grained metrics is to be enabled only upon the detection of an anomaly, this was considered acceptable.

FIELD NAME	FIELD DESCRIPTION
`fdn`	A *string* containing the sub-network, node, and interface identifiers.
`neid`	A *string* containing the sub network and node identifiers.
`moClass`	A descriptor for the type of device/interface.
`granularity`	The length of time over which the measurement was performed (in seconds).
`software`	The software and version running on the node.
`timestamp`	The time at which the measurement was taken (the end of the sampling interval).
`counters`	A variety of network metrics, calculated over the sampling interval, stored as a map with the counters' names as keys and their registered amount as values.
`pdfCounters`	Same as the `counters` field, and if one of the two contains information, the other is always empty (likely the result of naming differences during the collection and aggregation of this data).
`runid`	Always set to `datastream`. An artifact produced by the storage method used for the raw data.

Table 2.1: Description of the fields in the raw data set's schema.

2.5.1. Data Point Definition

To limit the scope and variety of data analyzed, the physical ports belonging to *routers* were chosen as the main focus of this thesis from among the numerous network nodes and interfaces present in the data set. When considering only these elements, the data set contains information from 12241 distinct interfaces belonging to 2245 different routers, with the number fluctuating between different sampling timestamps as a consequence of the missing data previously described.

A subset of nine metrics of interest for anomaly detection purposes was identified among the variable set of those appearing in the `counters` field of the selected interfaces. These counters were chosen based on several different aspects. Firstly, these metrics appear in the `counters` of more than 80% of router interfaces, ensuring that they contain meaningful information for a majority of the analyzed data. Secondly, these metrics were determined to contain useful information for identifying mobile network traffic anomalies by domain experts within the company. Finally, these metrics were deemed to be among those conventionally sampled from network nodes in the company's existing monitoring software, ensuring that they would not require the collection of additional data to be

METRIC NAME	METRIC DESCRIPTION
`inoctets`	The number of ingress *octets* (bytes of data) received, corresponding to the number of ingress *frames* received, excluding error frames.
`outoctets`	The number of egress *octets* (bytes of data) sent, corresponding to the number of egress frames sent.
`inpackets`	Incoming IP packets that need to be decrypted, counted by packets.
`outpackets`	Outgoing IP packets that need to be encrypted, counted by packets.
`ifInDiscardPkts`	Number ingress discarded IP packets.
`ifOutDiscardPkts`	Number of egress discarded IP packets.
`ifInErrorPkts`	Number of ingress error packets.
`ifOutErrorPkts`	Number of egress error packets.
`CrcErrorPkts`	The total number of CRC (cyclic redundancy check) error packets.

Table 2.2: Description of the selected counter metrics.

tracked in a production setting, which would increase the load on the network. The nine selected counters and their explanation are listed in Table 2.2.

Each interface is thus characterized by a 15-minute granularity multivariate time series of which the different counters are the features, and an example for one month of data can be observed in Figure 2.5.

Upon visual inspection, a large number of the selected time series appear to be characterized by a discernible seasonality in the first four counters, and display minimal trend over the span of the six sampled months. It is nevertheless possible that more significant trends and patterns may be observed when observing a larger period of time than that present in the data set, such as multiple years.

In summary, a data point for the anomaly detection process can be defined as:

DATA POINT

A set of nine network metrics calculated over a 15-minute time window from a single interface of one network element.

For each of these data points, the anomaly detection system should provide an anomaly

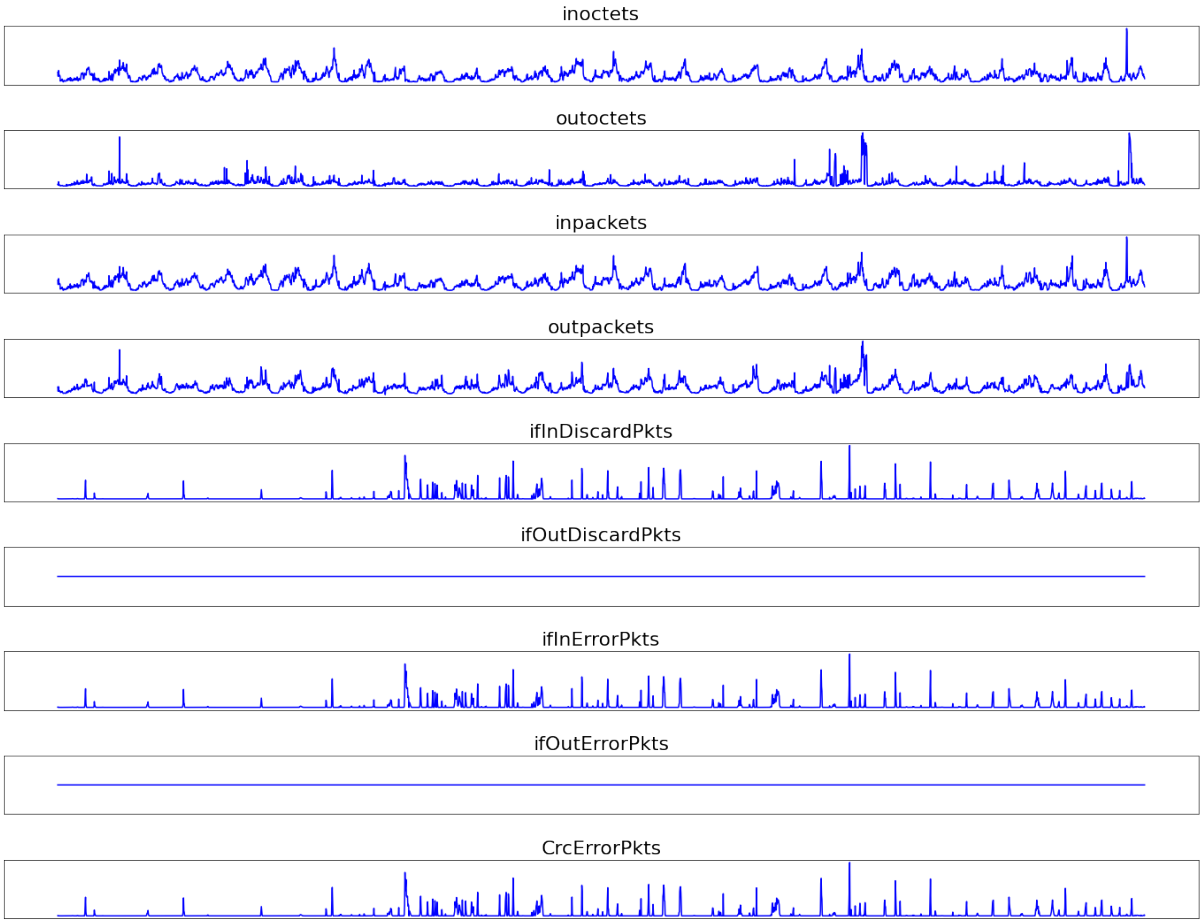


Figure 2.5: Example of one interface's time series across one month of samples. The x-axis represents time with 15-minute granularity over the course of July 2022. The y-axis represents the value of the metric whose name is indicated above the plot.

label, indicating whether or not the observation is determined to be anomalous.

2.6. Computational Resources and Tools

Because the data used to conduct this thesis was drawn from a real-world network belonging to *Ericsson*'s customers, the company's data governance and security policies imposed constraints on where and how the data could be stored and utilized in order to avoid the exposure of sensible information. Among other things, these policies required all computer code utilizing the data set to be run on a cloud-based `JupyterLab` [32] environment, connected to a *data lake* in which the data set was stored in the form of `pickle`, `parquet`, and `csv` files. Since this developing environment was intended as a shared resource for data science experimentation within the company, the computational resources available to each user were severely limited, which imposed constraints on the kinds of

computations that could be conducted during some of the thesis work. Most notably, the absence of a GPU made experimentation with modern deep learning techniques particularly impractical, as training such models would require prohibitive amounts of time The resources available through the computation environment are reported in detail in Table 2.3.

RESOURCE	VALUE
CPU Model	8x Intel Xeon Processor (Cascadelake) @ 2.30 GHz
CPU Cache	16 MB
CPU Cores	1
RAM	14 GB
GPU Model	None

Table 2.3: Computational environment hardware specifications.

The implementation of this thesis was carried out using version 3.8.10 of the `Python` programming language. [53] Table 2.4 lists the version of the software libraries used throughout this work.

LIBRARY	VERSION
`gluonts` [2]	0.13.1
`jupyterlab` [32]	3.2.6
`kats` [28]	0.2.0
`pandas` [39] [56]	1.5.3
`pyspark` [61]	3.1.1
`quidditch` [23]	1.4.0
`salesforce-merlion` [6]	2.0.2
`scikit-learn` [41]	1.2.2
`statsmodels` [49]	0.14.0
`tslearn` [50]	0.5.3.2

Table 2.4: Software libraries used during the course of this thesis.

2.7. Solution Architecture

To create an Anomaly Detection System compatible with the company's MSA-based SDN solution, the system was designed to be split into several separate microservices, as

described in Figure 2.6.

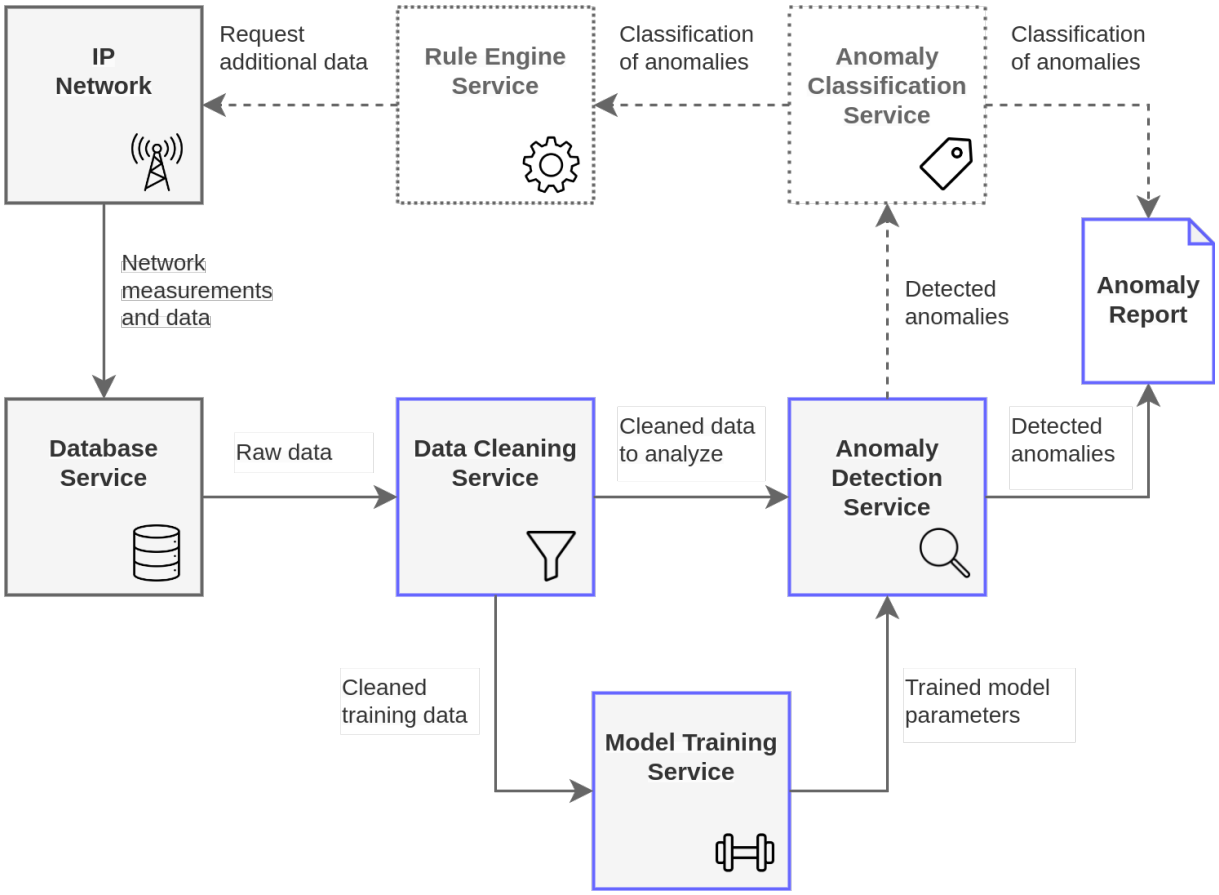


Figure 2.6: High-level architecture of the Anomaly Detection System. Elements with a gray border are existing *Ericsson* infrastructure. Elements with a blue border are the object of the implementation of this thesis. Elements with a dotted border are planned expansions of the system to be carried out during future work.

As stated in Section 1.1, the goal of the system is to autonomously identify abnormalities in the network data, to allow the use of automated data collection protocols in response to the detected anomalies, as well as to assist network operators during maintenance and monitoring. The proposed system covers all of the five stages of the network monitoring process described in Section 2.3, although the focus of the implementation of this thesis are exclusively the preprocessing, transmission, and analysis steps.

In this system, the raw data is extracted from the network nodes and stored in a general database (collection stage), from which other systems and services may draw the needed information. For the purpose of this ADS, the data is then queried by a *data cleaning service*, whose purpose is to reshape the data to prepare it for anomaly detection (preprocessing stage). The data cleaning service stores the data it produces in an

apposite database (transmission stage), where it can be extracted by other services when needed. This allows each data point to be processed a single time, instead of requiring the computations to be repeated whenever the data is needed.

A subset of the cleaned data is extracted at regular intervals by a *model training service*, which trains one or more anomaly detection models on a desired amount of recent observations. This ensures that the system is always up to date with changes occurring within the data.

The most recent data that has not yet been used in training is then taken as input by an *anomaly detection service* with a configured prediction frequency (analysis stage). This service is entirely dedicated to identifying anomalous samples within the data, for example by using the models trained by the training service. The anomaly detection service outputs an *anomaly report*, a data set in which each analyzed sample is labeled as either anomalous or not anomalous. For those outliers whose abnormality cannot be determined with absolute certainty, an anomaly score indicating the system's confidence in the prediction is also provided (presentation stage).

As an extension to this process, the system was designed to be expanded upon through the addition of an *anomaly classification service*, which would take the output of the detection service and attempt to classify the identified anomalies into different categories. The output of the anomaly detection service, integrated with that of the classification service, could then be fed into a *rule engine service* which would execute automated actions set up by the system's operator in response to the received information. This would include providing alerts to the operator when certain conditions are met, such as a certain number of anomalies occurring on the same network element in a short period, and requesting the retrieval of more detailed monitoring information from affected nodes. The additional information could then be used by the network operator for diagnostic purposes, or fed to other automated systems to provide a more precise classification of the anomalies or root cause analyses.

2.7.1. System Specifications

For the development of the ADS, a number of specifications were defined to describe the desirable properties that such a system should possess to accomplish its stated goal. These specifications are not to be intended as strict requirements, as this thesis work aims to determine how such a system could be constructed, and how it would perform on real-life data They should rather be interpreted as principles to be followed during the decision processes involved in its implementation. These specifications are reported in Table 2.5.

CODE	SPECIFICATION
S1	The system should identify the majority of anomalies occurring within the data.
S2	The system should not mark a large number of ordinary observations as anomalous.
S3	The system should be able to adapt to the appearance of novel behaviors in the monitored metrics.
S4	The system should provide online predictions within the span of one sampling interval.
S5	The system should require low amounts of processing resources.
S6	The system should be able to scale to handle varying amounts of input data.
S7	The system should be built with reliable, stable technologies.
S8	The system should be easy to maintain and expand upon in future work.

Table 2.5: Specifications for the development of the ADS.

Specification S1 describes the main purpose of the system, which is to identify anomalous occurrences in the monitored metrics. Specification S2 is necessary to ensure that the system may be of use in a practical scenario: if the ADS frequently mis-labels ordinary data points as anomalies its reliability will diminish, reducing its usefulness to the users and making it harder to set up automated responses based on its predictions. Specification S3 is required because of the sudden changes in behavioral patterns that network monitoring metrics can be subjected to, as a result of changes in the environment or network configuration. Specification S4 is motivated by the goal of establishing automated responses to the system's predictions, as the execution of some, such as requesting additional diagnostic data from the network, would require the anomalies to be detected as soon as they arise to provide the greatest benefits. Specification S5 is dictated by the desire to develop the system in the form of software to be run on client hardware, which offers no guarantees on the available processing power. Specification S6 aims to guarantee the system's ability to handle IP networks of varying sizes, to adapt to the customer's needs. Specifications S7 and S8, finally, have the objective of ensuring that the system is built according to the quality standards set by *Ericsson* software, and that it may be the object of continuous maintenance and improvement.

3 | Methods

The following chapter explains how the contents of Chapter 2 drove the choice of which techniques to adopt during the implementation of this thesis, which is detailed in Chapter 4. The chapter also provides a theoretical explanation for those techniques, and for the methods used to evaluate the performance of the system. Section 3.1 describes the data cleaning operations to be performed to prepare the data for use in anomaly detection. Section 3.2 describes some time series processing operations that were carried out during the implementation phase. Section 3.3 describes various concepts and techniques relating to clustering and, in particular, the clustering of time series. Section 3.4 describes the choice of anomaly detection algorithms and technologies used in the implementation phase, and provides a definition for the type of anomalies this work focuses on. Section 3.5, finally, describes the choice of how the evaluation of the tested detection algorithms should be conducted, and defines the metrics used to measure the models' performance.

3.1. Data Cleaning Methods

In order to prepare the raw data for ingestion by the anomaly detection service, various data cleaning operations should be performed.

Firstly, as discussed in Section 2.5.1, the data that does not pertain to physical router interfaces must be pruned out. The data should also be processed to eliminate unnecessary fields and optimize the ones that contain useful information to make their contents more readily accessible. In particular, the unique identifiers used to distinguish network elements are stored within the `fdn` and `neid` fields in the form of long and complex strings. Because accessing these identifiers requires parsing the text contained in those fields, their information should be extracted into individual fields. These identifiers should also be substituted with new, generated values to avoid the possibility of sensible information from the company's customers being leaked.

Furthermore, the time series features to be analyzed are contained in the `counters` field, a map of string keys and decimal values, which makes the extraction of data cumbersome

and contains, in most cases, numerous metrics that are not going to be utilized by the ADS. To facilitate the process of anomaly detection, the relevant information should be refined into distinct fields dedicated to each metric of interest.

Any discernible errors in the data must then be processed, such as mismatching field types, or data attributed to the wrong interfaces or timestamps. Finally, missing samples should be replaced with rows containing empty values in the feature fields (such as `null` or `NaN`), to enable them to be later imputed with artificial values to fill those gaps.

3.1.1. Missing Data Imputation

For the purpose of anomaly detection on a mobile transport network, missing data can be determined to be, itself, an anomaly to be reported, as it indicates a failure on some part of the data collection pipeline to provide the expected samples. Some time series algorithms, however, have diminished performance when a part of the data points are missing, or even fail to function at all. To negate this problem, several different data imputation techniques have been proposed.

When replacing missing values, the aim is to alter the original characteristics of the data as little as possible. This can be achieved by inserting values that are as similar as possible to what the true behavior of the time series would have been, if the data had been sampled correctly. Achieving this result, however, can be quite difficult, as it requires a good grasp of the fundamental process that generated the data in the first place.

This task is made harder when the time series in question presents strong trend or seasonality components, as filling the gaps with values that respect the sequence's global characteristics without taking into account those elements often fails to reconstruct the original behavior of the series. As discussed in Section 2.5.1, the time series that compose the data set for this study almost universally show strong seasonality components, which is the main complicating factor in their handling, whereas most interfaces show negligible trends on the monthly scale.

Among the various existing imputation techniques, the following ways to replace missing values were considered for implementation:

- **Fill with zeroes**: this approach is extremely simple to implement, but will invariably result in strong deviations from the original time series' characteristics (unless, of course, the sequence was solely composed of zero values).
- **Fill with series' mean**: this approach is quite simple to implement, requiring only the calculation of the global average of values, and maintains unaltered the

mean value of the data. It however alters the sequence's standard deviation significantly in the presence of series with changing values, especially those with trend and seasonality components.

- **Fill with previous valid value**: this approach is supported by many popular software libraries, and results in small deviations from the time series' normal characteristics when missing data points are few and separate from one another. When multiple data points are consecutively missing, however, it results in 'flat' sections that disregard trend and seasonality.
- **Fill through linear interpolation of previous and next valid values**: this approach is supported by a few common software libraries, and often has improved performance compared to previous approaches due to its ability to approximate local time series trend and behavior. It is nevertheless unable to preserve seasonality behavior when entire cycles are missing from the original series.
- **Fill with previous valid value sampled at the same day of the week, hour and minute**: this approach is of more difficult implementation than the previous, but allows to fill missing values while preserving the behavior that the series is known to take at a given temporal coordinate (assuming daily and weekly cycles are the prevailing factors). This can restore even consecutive missing parts of the time series while altering its characteristics very little and even maintaining seasonal behavior, but will fail to account for the presence of trends.
- **Fill with prediction from forecasting algorithm**: this is a complex approach that requires the use of a sufficiently well-performing time series forecasting model, which would be used to forecast the behavior of the series in missing sections, even when encompassing numerous cycles of seasonality. The efficacy of this approach is proportional to its ability to accurately model and predict the original time series' behavior and is more computationally burdening, but is capable of accounting for both seasonal and trend components.

3.2. Time Series Processing Methods

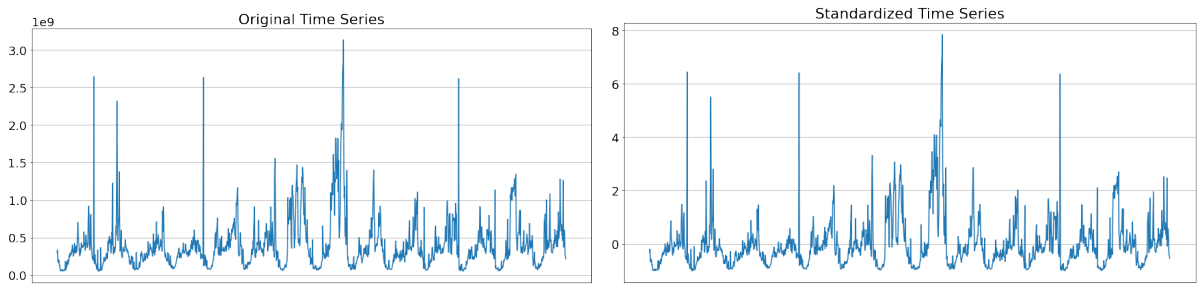
Numerous operations can be performed on time series to extract meaningful information regarding their characteristics and behavior, as well as to alter their properties to make them better suited for various tasks. Listed below are some techniques that were used at various points and for various purposes during the course of this thesis, so as to provide a reference to their execution.

3.2.1. Scaling

A common pre-processing step for numerous time series tasks is that of scaling, which aims to bring sequences that take different ranges of values to a common, comparable scale. One example of scaling operation is that of standardization. A data set is called standardized if it has a mean μ of zero and a standard deviation σ of one. Thus, given an univariate time series T whose global mean and standard deviation are μ and σ respectively, each sample x of T can be standardized by computing:

$$x' = \frac{x - \mu}{\sigma} \quad , \quad \forall x \in T$$

When dealing with a multivariate time series this operation can be computed on each variable independently, so long as preserving the relationship in scale between the different features is not important. Standardization provides a good way to bring different time series to similar scales while at the same time being resistant to outliers, meaning that the same time series, with and without outliers, should be brought to an approximately comparable scaling if data points with outlying values are a small percentage of the total. A comparison between the unscaled and standardized versions of a time series is provided in Figure 3.1 as an example.



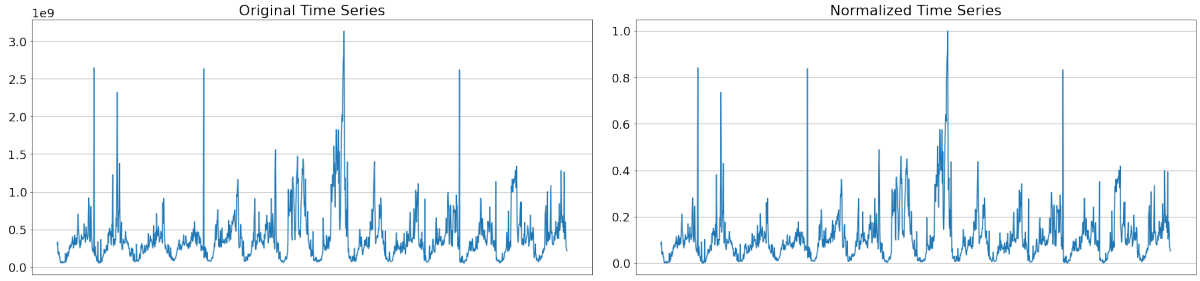
(a) Unscaled time series. (b) Standardized time series.

Figure 3.1: Comparison between an unscaled and standardized time series. The x-axis represents time, while the y-axis indicates the value of the time series.

A similar operation to standardization is that of normalization. A data set is called normalized if all of its values x are contained within the interval $[0, 1]$, with the maximum and minimum values being one and zero respectively. Thus, given an univariate time series T whose global maximum and minimum are x_{max} and x_{min} respectively, each sample x of T can be normalized by computing:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \quad , \quad \forall x \in T$$

When dealing with a multivariate time series this operation can once again be computed on each variable independently, so long as preserving the relationship in scale between the features is not important. Normalization is useful for bringing time series to comparable scales when the scaled values must fall within a specified range without exception. A comparison between the unscaled and normalized versions of a time series is provided in Figure 3.2, and a comparison between the scaling provided by standardization and normalization is provided in Figure 3.3.



(a) Unscaled time series. (b) Normalized time series.

Figure 3.2: Comparison between an unscaled and normalized time series. The x-axis represents time, while the y-axis indicates the value of the time series.

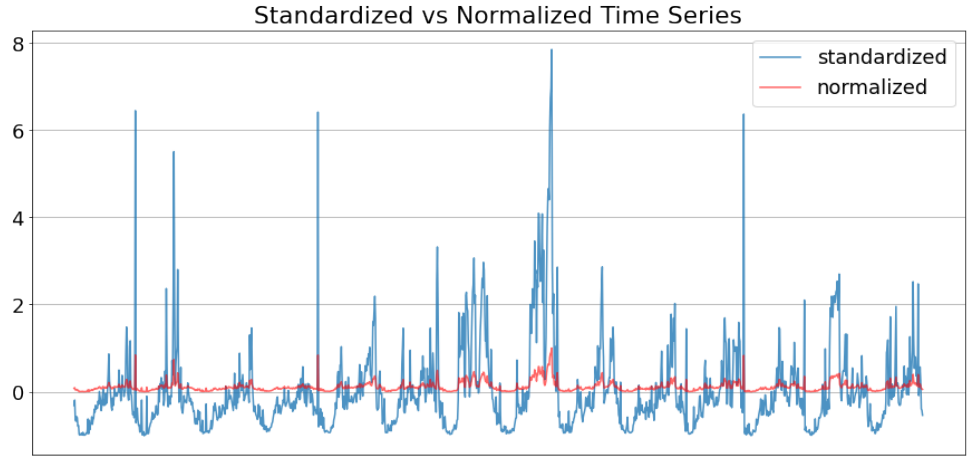


Figure 3.3: Scale comparison between standardized and normalized time series. The x-axis represents time, while the y-axis indicates the value of the time series.

3.2.2. Seasonal-Trend Decomposition

The presence of the trend or seasonality components described in Section 2.4.1 can make analyzing, forecasting, and performing anomaly detection on time series a complex endeavor. One popular tool for quantifying the effect of these factors is Seasonal-Trend

decomposition based on Loess (STL). [17] STL is a technique that relies on the smoothing provided by a regression curve to aid in decomposing a seasonal time series Y into three components: a trend component T, a seasonality component S, and a remainder component R accounting for the residual noise. As such, this technique assumes that for each data point v in the original time series:

$$Y_v = T_v + S_v + R_v$$

The residual component R describes the remaining variation in the data after seasonality and trend are removed, and can be used as a general indication of when the data is diverging from its expected behavior. An example of STL decomposition being applied to the `inoctets` feature of a time series from the available data set is provided in Figure 3.4.

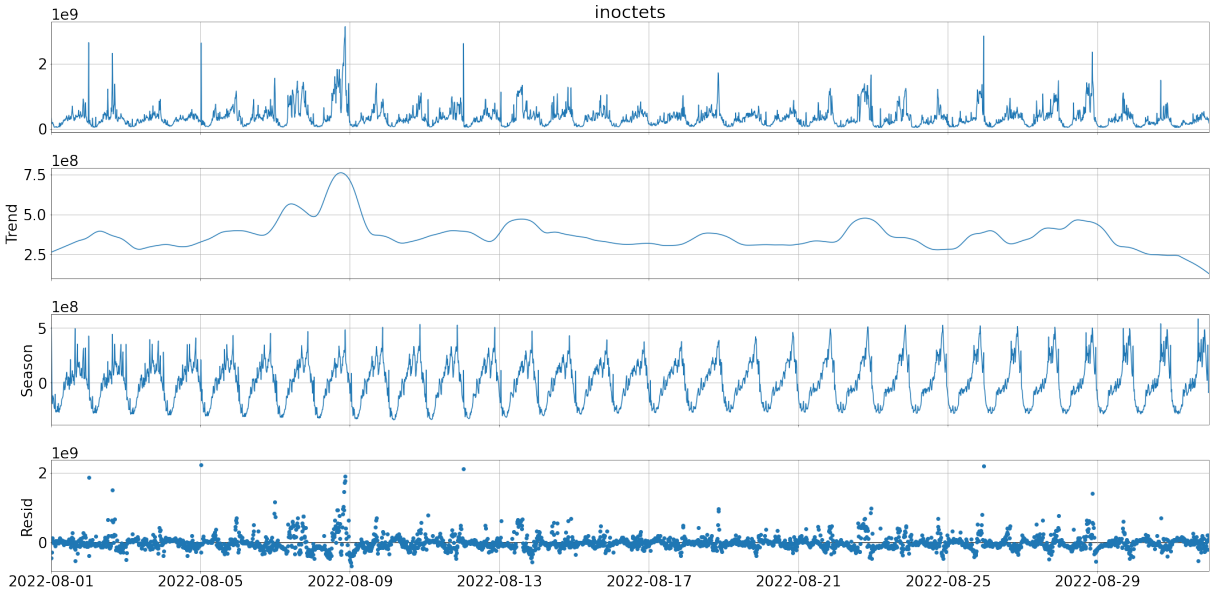


Figure 3.4: STL decomposition applied to an example time series. From top to bottom, the figure displays the original time series, its trend component, its seasonal component, and its remainder component. The x-axis represents time, while the y-axes represent the value of each component.

3.2.3. Principal Component Analysis

One common way to convert a time series into a feature vector is to employ Principal Component Analysis (PCA) [40], an algorithm commonly used for dimensionality reduction. PCA aims to detect structures in the relationship between features by identifying variables (principal components) that, when orthogonally combined, explain the majority of the variability in the original data. [35] Choosing an increasing number of principal

components thus makes it possible to retain a greater amount of information through the transformation, at the expense of the transformed data having a larger number of features. PCA can be performed to obtain an alternative representation of the data, in order to use algorithms and techniques that do not take the temporal aspect of time series into account.

3.3. Clustering Methods

The data set contains a very large number of different time series, making it difficult to visualize the full variety of behavioral patterns that the network interfaces can exhibit. One way to explore the data set more efficiently is to perform clustering on its time series, in order to group them by some measure of similarity and make it possible to visualize its variety more succinctly.

Warren Liao [55] defines clustering as the process of identifying structure within unlabeled data, by organizing it into groups such that the distance between objects in different groups is maximized, and that the one between objects in the same groups is minimized. Numerous algorithms have been created for the specific purpose of time series clustering, for the most part by either adapting existing approaches for static data to sequential data, or by converting time series into a static form and then applying the former approaches directly. In the first case, the main alteration lies in using a distance or similarity measure specifically suited for time series data, whereas in the second case, the data is first converted into either a feature vector or several model parameters (for example through PCA), on which conventional clustering is then applied. [55]

3.3.1. Clustering Algorithms

A large number of different clustering approaches have been developed for a variety of different applications. From among this varied and extensive set, two were experimented with during the course of this thesis: K-Means and DBSCAN.

- **K-Means** [37] is a well-known algorithm for data clustering, where every group's centroid is identified by the average value of the objects that are inside of it. The algorithm works by minimizing an objective function, usually determined to be some measure of the total distance between the objects in the data and their respective cluster's centers. A solution is reached by iterating between distributing the objects in the data set into the most appropriate cluster (initialized to an arbitrary position), and updating the centroid of the cluster by averaging its members' coordinates. The

two steps are repeated until convergence, attained when the value of the objective function cannot be reduced anymore. [55] This algorithm requires the choice of the parameter k, which indicates the number of clusters, and has a tendency to produce clusters of round shape.

- **DBSCAN** [21] is another popular approach for data clustering which groups together points that are densely packed together. The algorithm operates by choosing a random object, and calculating the number of objects in its vicinity by using an appropriate distance function and a threshold ϵ. A new cluster is formed if an object has a sufficiently large number of neighbors, and the operation is then repeated for those points, expanding the cluster until the minimum number of neighbors is not reached by the objects at the cluster's edge. At the end of this process, all of the densely packed objects are gathered together in clusters, with any leftovers being labeled as noise. This algorithm has the advantage of not requiring the prior definition of the number of clusters to be formed, and is capable of identifying arbitrarily-shaped clusters.

3.3.2. Dynamic Time Warping

A similarity measure for sequential data, necessary to apply most clustering techniques to time series without transforming them, is harder to define than those for static data because the order of data points in the sequence must be taken into account. [42]

A similarity measure based on Dynamic Programming commonly referred to as Dynamic Time Warping (DTW) has been initially proposed for speech recognition [47] tasks, and has since been adopted for a variety of applications. DTW is capable of identifying the optimal alignment between two time series, as well as providing the similarity between them in the form of their alignment cost. Figure 3.5, taken from *Information Retrieval for Music and Motion*, by Meinard Müller. [38], provides an example of time alignment using DTW, with the arrows indicating aligned points in the two sequences.

Petitjean et al. [42] propose an averaging method for a set of sequences under DTW called DTW Barycenter Averaging (DBA), which is designed to be used in similarity-based clustering techniques, especially those that require averaging such as K-Means. Further work proves that it can be used to improve the performance and accuracy of time series classifiers. [43]

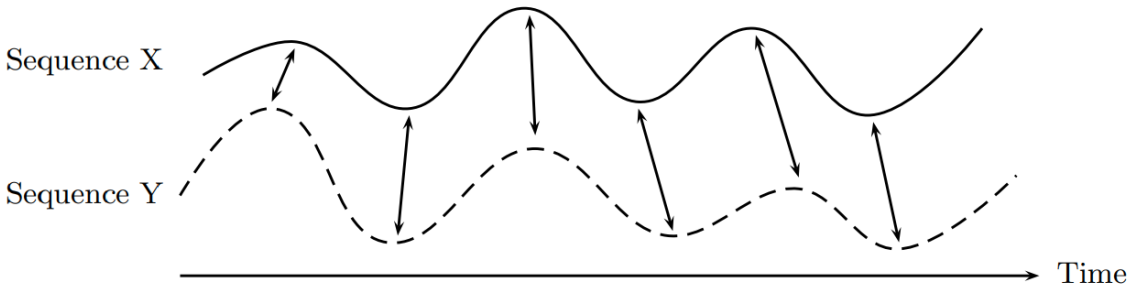


Figure 3.5: Example of alignment of two sequences using DTW.

3.3.3. Clustering Evaluation

Evaluating the results of a clustering process can be difficult when the optimal grouping is not known beforehand, as each algorithm can have various tunable parameters. To provide a measure of the quality of a specific clustering attempt, a large number of different measures have been proposed. Among these, the silhouette score [46] is a common method used to quantify the degree of consistency of clustered data. It offers a condensed measure of how similar an object is to those in its own cluster compared to those of different ones, and assumes values between -1 and +1, with higher values indicating more coherence.

The silhouette score on its own, however, can be insufficient to determine the appropriate value to attribute to a parameter, depending on the objective of the clustering process. The elbow method is a simple heuristic that is often used to determine the best value to be attributed to a parameter during clustering. The method entails plotting a graph of the variation of the data explained by clustering using each number of clusters, in order to then select the elbow of the curve as the value to use for the parameter. The purpose of this choice is to find the point where increasing the parameter will provide diminishing returns. The variation in the data can be calculated in many ways, including using the silhouette score. The elbow method is rightly criticized for its subjective and approximate nature, but also provides an evaluation strategy that is quickly and easily implemented. Because clustering is not the main focus of this thesis, the use of the elbow method was determined to be an acceptable compromise to expedite the process.

3.4. Anomaly Detection Methods

As stated in Section 2.5, the data set used in this work does not contain any labeled anomalies. This is a common occurrence when dealing with a large number of time series sampled at a high frequency, and by discussing the issue with company experts it was

deemed unfeasible for labeled data to be provided to such a system when deployed in a real-world environment. As a result, only unsupervised anomaly detection approaches were considered for the development of the ADS. An important consequence of this fact is the absence of an objective, quantitative, and automated way to evaluate changes in the system's detection performance at run time, once deployed.

Several approaches to the implementation of the anomaly detection process were considered, and the choice on which technique to adopt was dictated in accordance to the available resources described in Section 2.6 and the specifications detailed in Section 2.7.1. The three main options were determined to be the creation of a single, global anomaly detection model, the creation of multiple models each dedicated to a subset of similar time series, or the creation of a different model for each interface or time series feature. The results of these considerations are as follows:

- **Global Model**: such a model would be initially trained on a large and varied training data set, and be tasked with detecting anomalies in all monitored interfaces. This would have the advantage of requiring the ADS to load a single model from memory, reducing the time required to access memory storage, and would need to be retrained only occasionally to account for large changes in the metrics' behavior. On the other hand, training the model would require vast amounts of computational resources, which lies in opposition to specification S5, and would require training to be carried out on *Ericsson* premises. This, however, would make it difficult to train the model to adapt to novel time series patterns as required by specification S3, because the new data would have to be obtained from customers. In addition, a new model would likely have to be created for each metric, or combination of metrics, chosen by customers for monitoring, if their behavior differs significantly. Finally, the limited resources available during this thesis would make the development of such a system quite difficult, as it would likely have to rely on deep learning technology which is prohibitively slow to train in the absence of GPUs.
- **Subset-Specific Models**: this approach would require the creation of a small number of different models, each specialized in dealing with a distinct subset of interfaces. These sets of interfaces would be grouped based on similarities in their behavior, for example through clustering. This method would have the advantage of creating models specific to each behavioral pattern, thereby allowing the use of less powerful algorithms that could be trained on customer premises, all the while maintaining the total number of models relatively low. The appearance of new behaviors or the addition of new metrics would also be made easier, such as by automating a process that generates a new cluster and a new, dedicated model. It would, however,

require performing clustering on the interface's time series at regular intervals (to ensure that each interface is assigned to the correct model, and each model trained on the correct time series) to account for changes in the time series' behavior. To achieve this, a sufficiently efficient method for univariate or multivariate time series clustering would have to be identified, as specification S4 requires the system to be able to provide predictions within one sampling interval, while also being able to reliably distinguish time series with different behaviors. Once again, finally, the limited resources available would make the process of performing thorough clustering experimentation extremely lengthy, as many such algorithms have elevated time complexities.

- **Interface/Feature-Specific Models**: this approach would entail the creation of a separate, lightweight model for each interface (multivariate techniques) or time series (univariate techniques), trained with the desired frequency to adapt to changes in their behavior. This approach allows for maximum flexibility to adapt to each customer's use case, while at the same ensuring that the models are extremely specialized to detect anomalies in their monitored metric(s). Unfortunately, this also results in a much greater number of models having to be created, trained, and loaded, which can result in a significant slowing of the system's operations. Thankfully, because the system is to be developed as a MSA-based solution, this process could easily be parallelized by replicating the anomaly detection service through a linear increase in computational resources.

In the end, it was chosen to proceed with development adopting the interface-specific and feature-specific model approaches, as they allowed for maximum versatility while requiring resources compatible with those available for experimentation.

Another decision to be made was the choice of technologies to adopt in the implementation of the anomaly detection algorithms. In accordance with specifications S6 and S7, the ADS should be easily maintainable, expansible, and extendable through future work, favoring technologies that may help ensure that the system is easy to modify and upgrade (for example through the addition of new functionalities). Furthermore, in the interest of being able to conduct a sufficient number of experiments in the limited time available, tools that could provide reliable access to a variety of different anomaly detection models and support tools were deemed desirable. As a result, the best course of action was determined to be the use of the `Python` programming language [53], because of its wide adoption in the industry and extensive ecosystem of relevant libraries. In particular, the `Python` language offers a number of open-source libraries specialized in time series anomaly detection, such as `Darts` [26], `Luminaire` [12], `Merlion` [6], `Kats` [28],

and `TODS` [34], as well as `Quidditch` [23], an inner-source library developed internally at *Ericsson*. Among these, `Merlion`, `Kats`, and `Quidditch` were chosen for experimentation, as they appeared to display many desirable characteristics, such as a wide range of anomaly detection algorithms corresponding to the characteristics listed previously, integrated auxiliary time series processing and visualization capabilities, and support by the open source and *Ericsson* community, respectively.

3.4.1. Anomaly Detection Models

A variety of different models were tested from each of the three chosen anomaly detection libraries, in order to evaluate the suitability of the library on a technical level and the performance of the models on the available data. These models are implementations of a variety of different univariate and multivariate approaches, including both statistical, machine learning, and deep learning methods.

The anomaly detection algorithms tested from the `Quidditch` library are time series implementations of the following techniques:

- **Auto Encoder** [8] (multivariate): a neural network approach for generating lower-dimensional representations of data unsupervisedly. Anomaly scores are determined by calculating the reconstruction error for each sample.
- **HDBSCAN** [11] (multivariate): a density-based, hierarchical clustering method. It labels low-density points as anomalies based on their distance to the nearest cluster.
- **Isolation Forest** [36] (multivariate): a method that partitions the data into trees, identifying anomalies as those points requiring fewer partitions to be isolated.
- **KnnCAD** [10] (univariate): a density-based and distance-based method that flags as anomalies points whose distance with their k-nearest neighbors is significantly different from the rest of the data.
- **One Class SVM** [48] (multivariate): a classification approach that constructs a hyperplane enclosing the majority of data points, labeling as anomalies those left outside based on their distance from the plane.
- **Random Cut Forest** [24] (multivariate): an approach that creates a forest of randomly generated, equally deep trees, and identifies anomalies as those having on average shorter paths to be reached.

From among the `Kats` library, the following models were tested:

- **Outlier Detector** (univariate): a method similar to the one based on STL de-

composition described in Section 4.4.1, decomposes the time series and classifies as anomalies those points lying outside a specified interquartile range. Well suited for the detection of spike anomalies.

- **CUSUM Detector** (univariate): an algorithm designed to detect changes in the time series' mean and determine their anomalous nature based on hypothesis testing. It is specialized in detecting level shifts and changepoints.
- **Robust Stat Detector** (univariate): a method that works by smoothing the time series and highlighting its spikes through differentiation. The z-scores of the resulting values are used to determine anomalies. Suitable for detecting spikes and level shifts.
- **Multivariate Anomaly Detector** (multivariate): an algorithm that calculates each point's deviation from its expected value, computed using an appropriately fitted VAR (Vector Autoregressive) model that generates one-step-ahead forecasts of the time series. Suited for detecting anomalies in the relationship between different features of a multivariate time series.
- **Prophet Detector** (univariate): an anomaly detector based on the *Facebook* Prophet [51] forecasting model, which fits an additive model with various components such as trend, seasonality, and holidays to the data. Samples that strongly deviate from the forecast are classified as anomalies. Suitable for detecting any kind of anomaly on time series with strong seasonality.

Finally, the following `Merlion` library models were experimented with:

- **DBL - Dynamic Baseline** (univariate): a model that detects anomalies by calculating each sample's deviation from the behavior of historical data that has occurred in the same window of time, as defined by any combination of day, day of the week, or day of the month.
- **WindStats** (univariate): an anomaly detector based on the calculation of sliding window statistics, which assumes weekly seasonality in the data. It divides the week into buckets of a specified number of minutes, and determines anomaly scores for samples by comparing their value against those of historical data.
- **Spectral Residual** (univariate): an implementation of the algorithm described by Ren et al. [45], which combines the use of saliency maps and convolutional neural networks to extract anomalies from the frequency spectrum of the time series.
- **Stat Treshold** (univariate): a basic model that simply applies a provided thresh-

olding technique to the input data to determine anomalous samples.

- **ZMS - Multiple Z-Scores** (univariate): an algorithm specialized in detecting spikes and sharp trend changes, as well as their direction. It functions by building models of normalcy at multiple, exponentially growing time scales.
- **Isolation Forest** (multivariate): implementation of the algorithm proposed by Liu et al. [36] It partitions the data into trees, and identifies anomalies as those points requiring fewer partitions to be isolated.
- **Random Cut Forest** (multivariate): a refinement of Isolation Forest proposed by Guha et al. [24] It identifies anomalies by creating a forest of randomly generated trees that partition the data into smaller subsets, assigning larger anomaly scores to samples that are on average reachable through shorter paths.
- **Autoencoder** (multivariate): an algorithm that utilizes an autoencoder [3] to infer the correlations between different variables in a multivariate time series by compressing it into a lower-dimensional representation, or latent vector. By attempting to reconstruct the input and comparing the result with the original data, points with a significant reconstruction error can be classified as anomalies, as they deviate from the main patterns of the time series.
- **VAE - Variational Autoencoder** (multivariate): an improvement of the classic autoencoder approach proposed by Kingma and Welling. [31] In addition to the traditional encoder-decoder structure, this approach includes a probabilistic layer that models the latent space as a probability distribution. This allows to generate new samples by drawing from the learned distribution.
- **DAGMM - Deep Autoencoding Gaussian Mixture Model** (multivariate): an improvement of the classic autoencoder approach proposed by Zong et al. [62] This approach assumes that the latent space can be represented by a mixture of Gaussian distributions, which allows for the modeling of both normal and anomalous patterns. Anomaly scores are calculated based on both the reconstruction error and the log-likelihood of the occurrence of the data based on the estimated Gaussian Mixture Model.
- **LSTM ED - Long Short-Term Memory Encoder-Decoder** (multivariate): a model based on the classic encoder-decoder architecture where both networks are Long Short-Term Memory neural networks. The reconstruction error is used to estimate anomaly scores.
- **Prophet** (univariate): an anomaly detector based on *Facebook*'s Prophet model

for time series forecasting. [51] It fits an additive model with various components such as trend, seasonality, and holidays to the data, and assigns anomaly scores to observations based on their deviation from the model's forecasts.

3.4.2. Types of Anomalies of Interest

In order to experiment with different kinds of anomaly detection techniques and evaluate their performance, it is necessary to determine what type of behavior in the monitored data is considered an anomaly of interest for the ADS. It was determined that, for the purpose of this study, the system would be interested in drawing attention to three kinds of anomalies:

- **Missing values**: these are instances in which the value for one or more of the monitored metrics is absent from a given sampling timestamp. It corresponds to a failure to record or transmit the measurement, and can be identified with absolute certainty.
- **Domain violations**: these are instances in which the value for one or more of the monitored metrics lies outside the defined domain range that the metric should take under normal circumstances. For example, a metric counting the number of bytes received by an interface should never fall below zero. These anomalies correspond to some kind of fault in the monitoring equipment or in a corruption of the data, and can once again be identified with certainty so long as the domain range is correctly defined.
- **Unexpected behavior**: these are instances in which the value of one or more of the monitored metrics deviates significantly from the pattern established by previous observations. These anomalies can correspond to a wide variety of different changes happening inside the monitored network, and often it is not possible to identify them with absolute confidence. It is to detect this kind of anomaly that advanced anomaly detection algorithms have to be employed.

From among the nine metrics chosen for anomaly detection described in Section 2.5, it was decided that all metrics should be analyzed for missing values and domain violations. Because the last five (`ifInDiscardPkts`, `ifOutDiscardPkts`, `ifInErrorPkts`, `ifOutErrorPkts`, and `CrcErrorPkts`) already measure the occurrence of undesirable events in the network, however, it was determined that their expected domain should be set to zero, and the detection of unexpected behavior should instead focus on the first four counters (`inoctets`, `outoctets`, `inpackets`, and `outpackets`). The system was nevertheless designed in such a way that it could flexibly adapt to the analysis of any set

of monitoring metrics, including the full set of the nine selected features.

For the purpose of testing and comparing the performance of different detection algorithms and focusing the experimentation efforts, a subset of behavioral anomalies of interest for this thesis was also determined from among the various classifications described in Section 2.4.

Firstly, because the available data covers a range of only 6 months, any patterns repeating with a periodicity greater than monthly could not be easily taken into account, and thus only deviations from the seasonality occurring below monthly scope are considered when detecting anomalies. This also excludes the detection of anomalies caused by sudden shifts in the trend component of time series, as no such anomalies could be identified in the limited time frame of the available data.

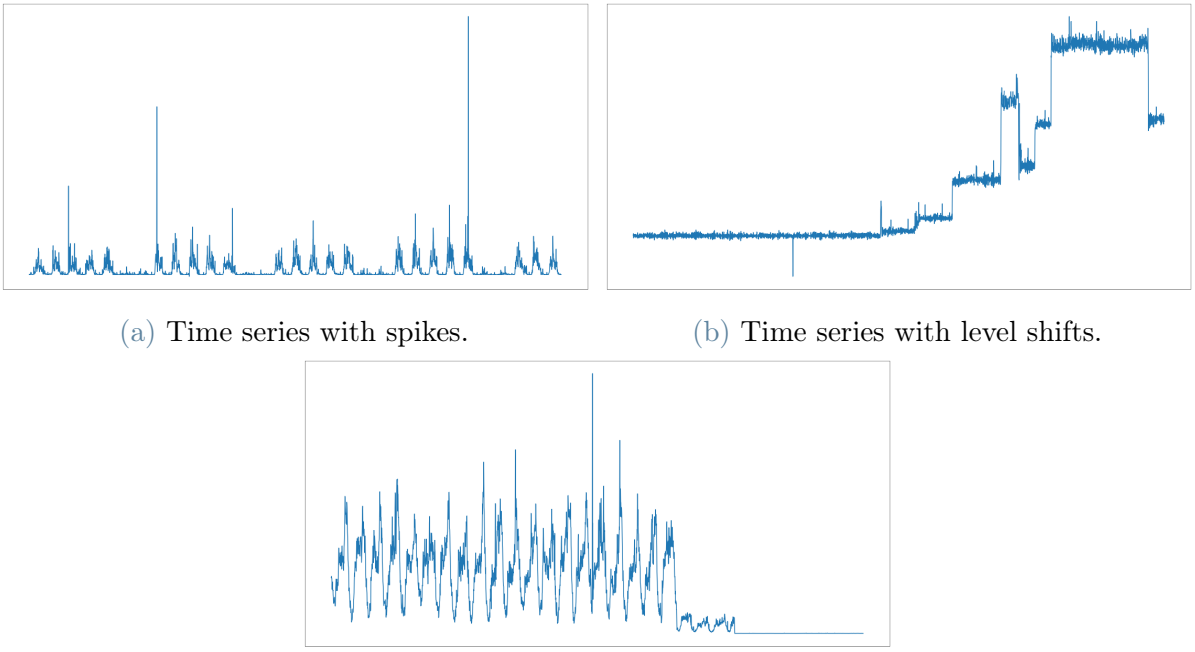
Secondly, this work focuses on detecting behavioral anomalies relating to individual observations, rather than entire sequences or time series. This is motivated by a variety of factors, including the desire to provide the ability to perform online anomaly detection, and the fact that the algorithms within the chosen libraries operate by providing labels or scores for each individual observation. These anomalies are nevertheless evaluated in the context of the surrounding data and the established pattern of the time series, and future work could aggregate predictions made for single samples into evaluations for longer sequences based on some defined parameters.

Finally, three main types of anomalous behaviors were identified to be occurring within the time series in the data set. While it is not of immediate interest for the system to be able to distinguish between these anomalies, it should nevertheless be able to detect all of these types. The description of each type is reported below, and examples for each are provided in Figure 3.6.

- **Spikes** (or dips): these are anomalies in which the value of the time series sharply increases or decreases in a manner that deviates from the sequence's usual seasonal pattern. The change lasts for a very small number of time steps, after which the series resumes its normal behavior.
- **Level Shifts**: these are anomalies in which the average value of the time series shifts upwards or downwards, but the behavior of the series remains otherwise unchanged. Shifts can be temporary, with the sequence returning to the original level after a certain amount of time, or permanent, with the new level becoming the normality for the time series going forward.
- **Changepoints**: these are anomalies in which the behavior of the time series changes

radically for a certain amount of time, potentially affecting both its mean, variance, and seasonality. The change might be permanent, or it might revert to the original behavior after some time has elapsed.

The three types of anomaly are not equally represented within the data. While computing precise statistics is made difficult by the absence of labeled outliers, it is apparent upon visual inspection that spike anomalies occur in the data with far greater frequency than the other categories.



(a) Time series with spikes.

(b) Time series with level shifts.

(c) Time series with changepoints.

Figure 3.6: Examples of the three types of anomalies of interest occurring within the data. The x-axis represents time, while the y-axis represents the value taken by the time series.

Anomalies in multivariate time series can also themselves be univariate or multivariate, depending on whether they affect one or more of the time series' features, and both univariate and multivariate detection techniques were experimented with during the course of this work.

3.5. Model Evaluation Methods

In order to identify which anomaly detection algorithm is best suited for use in the developed system, a way to compare their performance must be defined. However, because of the lack of labeled behavioral anomalies in the available data (see Section 2.5), this cannot be achieved by simply comparing the anomalies predicted by the system with a

set of ground-truth labels. Since this type of evaluation metric was deemed necessary to determine the capabilities of the models, three possible solutions were identified to solve the issue:

- **Gathering more data**: this solution would involve the collection of more real-life data, this time with marked anomalies that could be used for performance evaluation. While this method would be ideal for providing an indication of how the system would perform in production, collecting such data would require a significant amount of time, as well as for specific data collection software to be developed, and for a strategy to be devised to label the anomalies in the coming data. This would probably entail a mix of log analysis, cross-referencing with real-world events affecting the infrastructure and network traffic, and manual investigation of the time series, resulting in the use of a large number of company resources.
- **Manual labeling**: this solution would require manually analyzing a subset of time series from the data set, and creating a labeled set of evaluation time series where outlying points are indicated by an appropriate indicator. This approach would have the advantage of evaluating the system on authentic, real-world data, thereby ensuring that its performance is reflective of the one it would have when deployed on a live network. On the other hand, manually labeling the data points would require an extraordinary amount of time, which would severely limit the size of the evaluation data set produced. Furthermore, the labeling would heavily rely on the ability of the labeling agent to identify the anomalies in the data, thereby disproportionately favoring those outliers that are easily identifiable by the human eye, and exposing the process to high amounts of human bias and error.
- **Synthetic time series**: this solution would entail generating a new set of synthetic time series whose behavior resembles that of the original data if it were without anomalies, and then randomly injecting anomalies at various locations while generating a corresponding set of labels. By creating a process that can automatically generate new time series, this approach would allow to obtain a much larger evaluation data set in a smaller amount of time. Furthermore, because the anomalies would be injected randomly by the automated process, the resulting time series would present both hard and easy-to-identify labeled anomalies, reducing human bias. Another advantage of this approach is that, since the produced data is artificial, it would be possible to carry out the evaluation process in a different computational environment than the one used for experimentation, potentially enabling access to increased processing power. At the same time, however, the reliability of this method hinges on the fact that the synthetic time series and anomalies are

similar to those of the original set, but because of the inherent variation present in the data it is not possible to establish this fact with absolute certainty. Additionally, some of the original time series display highly irregular behavior, making it difficult to determine what parts are determined by outliers, and which are not.

After some consideration, it was decided that the generation of synthetic time series would be the best approach to adopt given the available time and computational resources. Because of this, however, it was decided to limit the evaluation of the models to univariate synthetic time series, as a suitable way to generate multivariate synthetic series that accurately reflect the original data could not be identified in the available time.

Several different metrics were considered to quantitatively evaluate the performance of the anomaly detection models. These evaluation metrics reflect the main performance indicators for the success of the system (see Section 2.7.1: the ability to correctly identify anomalies in the data while producing a low number of false alerts, hereby called detection accuracy (see specifications S1 and S2), and the amount of processing resources required for training the models, from here called computational requirements (see specifications S4 and S5).

3.5.1. Detection Accuracy Metrics

In classification problems, the terms *true positive*, *true negative*, *false positive* , and *false negative* are used to compare the predictions of the classifier to be evaluated with some trusted external labeling. To this purpose, the terms *positive* and *negative* refer to the prediction of the classifier, and the terms *true* and *false* indicate whether such prediction corresponds with the trusted labels. More specifically, in the field of anomaly detection (binary classification of observed data points as ordinary or anomalous):

- True positives (TP) are anomalous observations correctly flagged by the ADS.
- True negatives (TN) are ordinary observations correctly ignored by the ADS.
- False positives (FP) are ordinary observations mistakenly flagged by the ADS.
- False negatives (FN) are anomalous observations mistakenly ignored by the ADS.

From these definitions, the popular performance metrics *precision* and *recall* can be calculated. Precision can be seen as a measure of quality for the detector, indicating its ability to make correct predictions rather than mistaken ones. It can assume values between 0

and 1, with higher scores being more desirable, and is defined as:

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

Recall, on the other hand, can be seen as a measure of quantity, indicating which portion of the total anomalies the detector succeeds in identifying. It can assume values between 0 and 1, with higher scores being more desirable, and is defined as:

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

When used in isolation, precision and recall fail to offer a sufficiently complete picture of the algorithm's ability to identify anomalies. For example, an algorithm that flags every data point as anomalous would obtain a perfect recall score, and one that only makes a single prediction that happens to be correct, would result in a perfect precision score. Furthermore, under many circumstances, it is possible to increase one at the expense of the other, for example by raising or lowering the threshold applied to anomaly scores to sparsify predicted anomalies. To overcome these shortcomings, precision and recall are often reported alongside their harmonic mean, the F_1 score, which describes the system's accuracy by incorporating both of its components' contributions. It can assume values between 0 and 1, with higher scores being more desirable, and is defined as:

$$F_1 = 2 \cdot \frac{PRECISION \cdot RECALL}{PRECISION + RECALL}$$

Another similar metric to precision and recall is *fallout*. Fallout can be interpreted as a measure of the ability of the detector to avoid the misclassification of ordinary data points as anomalous. This metric is particularly relevant in the field of anomaly detection because a system that raises many false alarms may be perceived as unreliable by its users, and hinder the creation of automatic response systems based on its predictions. Fallout can assume values between 0 and 1, with lower scores being more desirable, and is defined as:

$$FALLOUT = \frac{FP}{FP + TN}$$

It is important to note that the detection accuracy metrics were calculated point-wise. This means that each anomaly is identified by a label marking a single observation as anomalous, and the correctness of the model's prediction is determined based on whether it detected that same sample as anomalous. In some use cases, however, it is of greater interest to the network operators to identify the general window of time in which the anomaly occurs rather than each specific point, as certain anomalies present unusual

behavior for extended periods of time (such as for level shifts and changepoints). For this purpose, *point-adjusted* [60] and *revised point-adjusted* [27] metrics have been proposed, which operate by treating anomalies as windows rather than labels tied to singular observations.

The decision was made to use point-wise metrics in this evaluation, however, because of the difficulty in determining an appropriate window size for anomalies, especially regarding permanent or protracted changes in the time series behavior. Furthermore, because some of the use cases for the system require online detection, its ability to detect anomalous behavior in individual new samples was deemed more relevant. It should nevertheless be taken into account that these factors cause a more pessimistic estimate of the model's performance to be calculated, as many algorithms have a tendency to label multiple observations as anomalous following the occurrence of level shifts and changepoints.

3.5.2. Anomaly-Specific Accuracy Metrics

A relevant aspect of the system's performance is its ability to identify all three different types of anomalies of interest described in Section 3.4.2. However, because the three groups are not equally represented within the data, evaluating the system without distinguishing between anomaly types would not produce a complete picture of its performance.

One possible solution to this problem is to evaluate the system on different synthetic data sets, each containing anomalies of a single type, and one additional one containing mixed anomalies. This, however, would require an amount of processing time four times greater than performing the evaluation on a single set, resulting in the necessity to use data sets of smaller sizes. Furthermore, since in the original data set level shift and changepoint anomalies almost always manifest in time series that also present spike anomalies, isolating those categories of outliers might result in distorted results.

As a result, the solution that was adopted was the additional computation of a partial *recall* metric for each type of anomaly. This metric, calculated by counting as *positives* only those labels pertaining to each specific kind of anomaly, makes it possible to determine the ability of the model to identify outliers of that specific type. Computing other metrics, such as precision, in a similar way would instead serve no purpose, as the model correctly identifying anomalies of other kinds in the data would be treated as a false positive, mistakenly penalizing it.

3.5.3. Computational Requirements Metrics

Because the system is to be deployed on customer hardware, the computational power involved in training and running its models is an important factor to consider. As general measures of the processing requirements of the model, the mean training time (MTT) and mean prediction time (MPT) of each algorithm were calculated. These quantities are defined as the average time required by the algorithm to be trained on a time series of 2976 samples (31 days), and to perform a prediction on one new sample, respectively. While these values are naturally affected by the hardware the models are deployed on, they still offer a useful measure of their performance relative to each other.

4 | Numerical Experiments

The following chapter explains how the techniques detailed in Chapter 3 were employed in the implementation of this thesis, with the results being discussed in Chapter 5. It describes the various experiments conducted throughout this work, and provides an evaluation of various tested anomaly detection models on the available data. Section 4.1 describes the specific cleaning operations performed on the data, as well as the output of that process. Section 4.2 describes the process of clustering the interfaces from the clean data through various techniques. Section 4.3 describes some time series analysis operations that were conducted on the cleaned data to better understand its characteristics. Section 4.4 describes the experiments conducted with different anomaly detection algorithms, their results, and the factors that lead to the choice of using a specific software library. Section 4.6 describes the evaluation of the selected anomaly detection models, and the parameters used in its execution. Section 4.7, finally, describes how the different elements of this chapter were brought together to produce the final anomaly detection system.

4.1. Data Cleaning Experiments

The first operation conducted as part of the implementation of the ADS was to perform several data cleaning operations on the available raw data, to make it more suitable for automated anomaly detection as described in Section 3.1. The sequence of transformations applied to the data is as follows:

1) **Interface selection**: the data not pertaining to the physical router interfaces of interest was filtered out of the data set. The network elements to be excluded from the data were identified by using the contents of the identification fields `fdn` and `neid`, as well as the `moClass` field.

2) **Identifier extraction**: the identifiers for the Network Node and Interface of each data point were extracted from the `fdn` and `neid` strings, and turned into separate columns, since they were revealed to be the only ones strictly required to uniquely

identify each interface. In doing so, some data that was mistakenly attributed to separate network elements despite originating from the same interface was unified under a single identifier, as it was determined to be the likely result of corruption in the data generation process.

3) **Data anonymization**: the identifiers extracted in the previous step were substituted with new, manually generated IDs. This was done to prevent the possibility of the data being connected to the real-life infrastructure that produced it, avoiding the leaking of private information from *Ericsson*'s clients.

4) **Counters extraction**: the subset of nine counters listed in 2.5 was unpacked from the `counters` field and turned into distinct columns. For those interfaces whose `counters` field was lacking one or more of those metrics, the corresponding column's value was set to zero.

5) **Pruning of excess fields**: all of the columns from the original data set were removed, with the sole exception of the `timestamp` field. The resulting schema for the cleaned data set is reported in Figure 4.1.

6) **Error correction**: all duplicate samples in the data (samples bearing the same identifiers and timestamp) were removed, as they contained the same data in all instances where such an overlap occurred. Furthermore, the first timestamp in each day (the timestamp for hours 00:00:00) was mistakenly attributed to the day preceding the actual sampling date, and this mistake was corrected for all such samples.

7) **Missing samples padding**: for each interface, all of the missing measurements were replaced by setting their value on all features to `NaN`. This was done to reduce the work required to substitute the missing values with the appropriate technique from those listed in 3.1.1, as required by each particular anomaly detection approach.

The data cleaning process was carried out using `PySpark` [61], a `Python` library that allows to perform large-scale data processing in a distributed environment. This choice was made to improve the efficiency of the cleaning operations during the implementation of the thesis, which was necessary because of the limitations on processing power described in Section 2.6. Furthermore, this ensured that the process could be easily transferred into its own *data cleaning microservice* as described in Section 2.7, as it can efficiently be scaled to handle smaller or larger loads of time series data.

```
root
 |-- node: string (nullable = true)
 |-- interface: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- inoctets: double (nullable = true)
 |-- outoctets: double (nullable = true)
 |-- inpackets: double (nullable = true)
 |-- outpackets: double (nullable = true)
 |-- ifInDiscardPkts: double (nullable = true)
 |-- ifOutDiscardPkts: double (nullable = true)
 |-- ifInErrorPkts: double (nullable = true)
 |-- ifOutErrorPkts: double (nullable = true)
 |-- CrcErrorPkts: double (nullable = true)
```

Figure 4.1: Schema of the cleaned data set, as provided by `PySpark`.

4.2. Clustering Experiments

The cleaned data set contains 12241 distinct multivariate time series, displaying various kinds of different behaviors and characteristics. To conduct anomaly detection experiments that would cover the full diversification of the data set within the time and resources available, it was necessary to create an experimental data set of smaller size that would still retain as much as possible of the variety expressed by the original data. To do so, the interfaces were grouped based on similarities in their time series using different clustering algorithms.

Both time series clustering approaches described in Section 3.3 were followed: applying a clustering algorithm adapted for time series to the data directly, and applying classic clustering algorithms to the data after transforming it using PCA (see Section 3.2.3. In both cases, any missing samples were imputed using the linear interpolation technique described in Section 3.1.1.

For the first approach, an implementation for time series of the K-Means algorithm from the `tslearn` [50] `Python` package was utilized, using DTW as a distance measure. Because of the elevated computational burden entailed by this technique, several further reductions had to be applied to the cleaned data to be able to perform the operation:

- Only a single month of data was used for clustering, with the choice falling on August 2022 as the month bearing the least amount of missing samples.

- Only a single of the four features of interest for anomaly detection was utilized (the `inoctets` metric, expressing the number of ingress bytes received by the interface).
- Only 10% of the total number of router interfaces was used. These were chosen at random from those that presented fewer than 20% of overall missing samples after the data cleaning operations were performed.

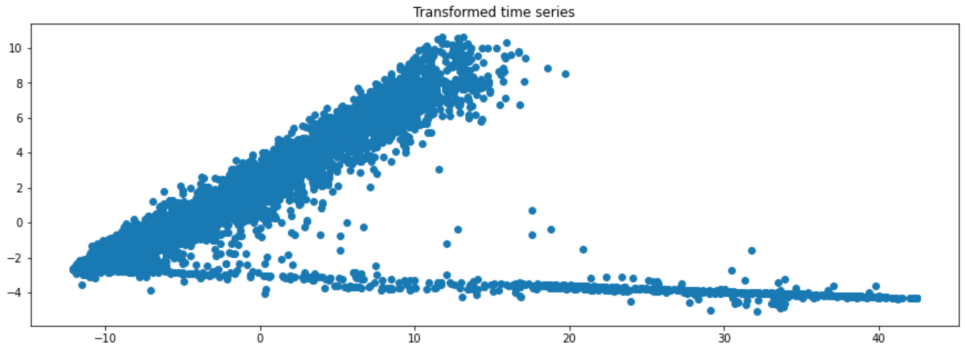
Because of the large amount of time required to execute this process even after applying the limitations listed above, only a small number of experiments was conducted. The clustering process was performed with both normalized and standardized time series (see Section 3.2.1), in order to compare the results obtained by either method, in both cases using 20 as the number of clusters. As the standardized set displayed more variety, one further clustering attempt was carried out using standardized time series and 30 clusters, to examine the consequences of increasing the number of groups. The two clustering operations displayed comparable results, with a similar ability to distinguish between time series with different behaviors. The results of this clustering approach are reported in Appendix A.1.

Given the severe limitations that had to be imposed on the number of interfaces used, however, it was decided to perform further experimentation using the second approach. This approach was carried out using the `scikit-learn` [41] `Python` library's implementations of the PCA and K-Means algorithms. In this case, the entirety of the data set was utilized, as the processing time required to carry out the clustering operation was significantly lower. The experimentation process was carried out as follows, both using normalized and standardized time series:

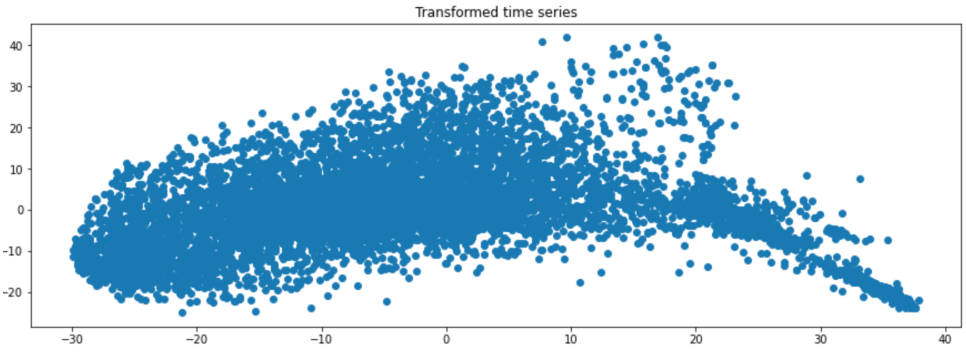
1) The data set was scaled using either normalization or standardization.
2) PCA was performed to investigate the cumulative variance explained by each number of principal components.
3) The scaled data was transformed through PCA using two principal components (for visualization purposes), as well as the number of principal components necessary to explain 95% of the scaled data set's variance (64 and 120 for the normalized and standardized data respectively).
4) The transformed data using two principal components was plotted.
5) The classic K-Means algorithm was repeatedly applied to the transformed data sets using a progressively increasing number of clusters, and the silhouette score for each attempt was computed.

6) Promising number of clusters were selected through the elbow method, and plotted to examine their results.

By visually comparing the plots of the transformed data with two principal components from both the standardized and normalized data sets, it can be observed that the data set scaled through standardization tends to form a much more homogeneous group within the new, lower-dimensional space. This is also reflected by the much greater number of principal components required to explain 95% or more of the standardized data set's variance. The results of these transformations are compared in Figure 4.2.



(a) Normalized time series.



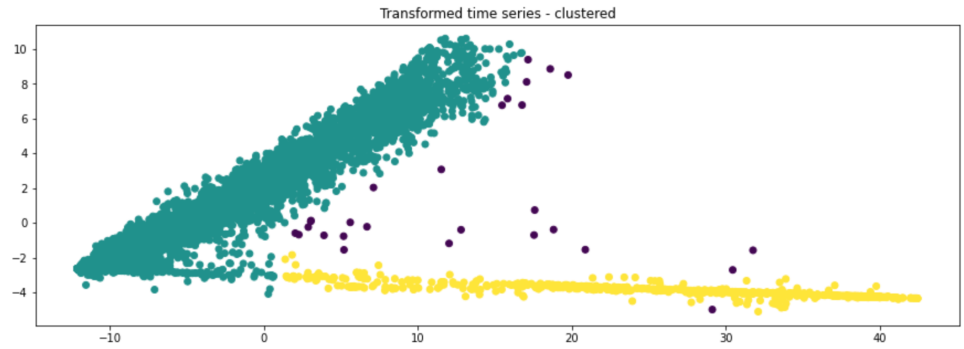
(b) Standardized time series.

Figure 4.2: Comparison between the normalized and standardized data sets transformed through PCA (two principal components). The x- and y-axes describe the first and second principal components, respectively.

The clustering results of the data transformed using two principal components were, predictably, unsatisfactory, as the variety displayed by the different clusters was very low. When using the number of principal components necessary to explain 95% of the variance, instead, the results were more promising, with the standardized data, in particular, displaying a more varied cluster set. The results of this clustering approach are reported in greater detail in Appendix A.2.

An additional experiment was carried out by applying the DBSCAN clustering algorithm to the data transformed by PCA using various numbers of principal components. In this case, the number of clusters is automatically determined by the algorithm, and the elbow method was instead applied to selecting the ϵ parameter. This process, however, produced less interesting results than those of the K-Means algorithm, as it would always result in a small number of clusters being formed, which failed to capture the desired time series variety. An example comparing the clustering performed by K-Means and DBSCAN on the data transformed using two principal components is provided in Figure 4.3, and some of the results are displayed in more detail in Appendix A.3.

(a) K-Means clustering.



(b) DBSCAN clustering.

Figure 4.3: K-Means and DBSCAN clustering comparison on the normalized data transformed using two principal components. The x- and y-axes describe the first and second principal components, respectively.

Ultimately, however, the results obtained from the PCA-based clustering were determined to be inferior to those of the first approach, which was able to capture a wider variety of significantly different time series. As a result, the data set to be used in anomaly detection experimentation was created by drawing one random member from each of the 20 clusters identified by the DTW-based method using standardized data. This data set, which will be referred to as the experimentation data set from this point on, is displayed

in Appendix B.1.

4.3. Time Series Decomposition Experiments

By visually inspecting the experimentation data set, it can be observed that most of the interfaces' time series are characterized by a strong daily seasonality, with some displaying seasonal behavior also on a weekly scale (with the weekend behaving differently from the weekdays) and on a sub-daily scale (with some sequences presenting two distinct peaks within the normal daily behavior).

Assuming that anomalies in the data with strong seasonality could be identifiable as significant deviations from the seasonal behavior of each time series, an attempt was made to remove the seasonal behavior from the time series. To accomplish this, the implementation of STL decomposition from the `statsmodels` [49] `Python` library was used, as described in Section 3.2.2.

The decomposition was performed on a single month of data of the first feature (`inoctets`) for each of the time series in the experimentation data set, attempting to isolate the seasonality occurring with daily frequency (or 96 samples). The time series was then reconstructed by adding only the trend and seasonality components, to simulate its behavior in the absence of anomalies and compare it with the original and the residual component. An example of the results of this process is displayed in Figure B.2, with the full results on the experimentation data set being reported in Appendix B.2.

The reconstructed signal still appears noisy in most cases, but seems to effectively smooth over most spike anomalies, which are then recognizable in the residual component's plot as outlying values compared to the global mean and variance of the component. When encountering level shift or changepoint anomalies, on the other hand, the reconstructed time series has a more unpredictable behavior, with the reconstruction's attempts to adapt to the changes in behavior leading to high variance in the residuals component.

Further attempts at decomposition were performed using `statsmodels`' MSTL algorithm, which applies STL decomposition using multiple seasonal components. Specifically, the seasonal components utilized were 48 samples (half a day), 96 samples (one day), and 672 samples (one week) in length. The results of this process, however, show little difference from those obtained using the regular STL decomposition algorithm.

In conclusion, the decomposition process showed promising results in highlighting outlying data points in the form of spike anomalies, but appears to become chaotic in the presence of level shifts or changepoints.

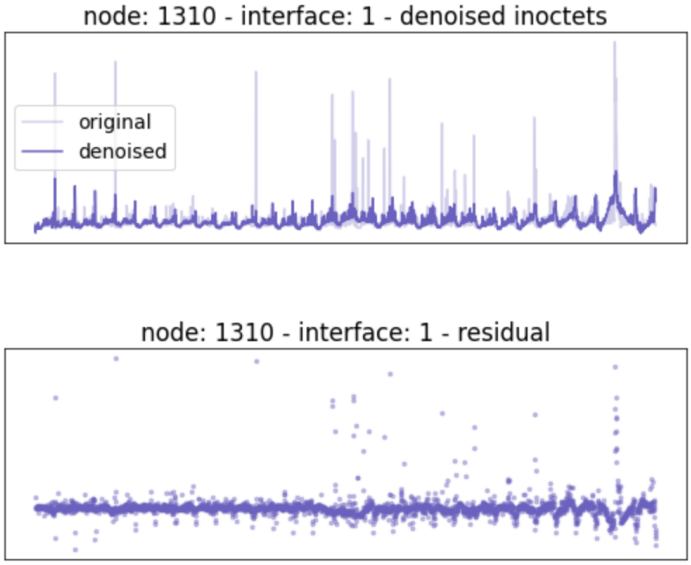


Figure 4.4: Results of STL decomposition and reconstruction for one time series feature. The x-axis represents time, while the y-axis represents the value of the original and reconstructed time series in the top image, and that of the residual component in the bottom image.

4.4. Anomaly Detection

Several anomaly detection techniques were applied to the experimentation data set obtained from the clustering process, in order to gauge their performance both in terms of processing time and accuracy in detecting anomalies according to specifications S1, S2, S3, and S5 listed in Section 2.7.1. Throughout this process, the tools from different anomaly detection libraries were also compared to determine which would be the most suitable to be used in the final ADS based on specifications S7 and S8.

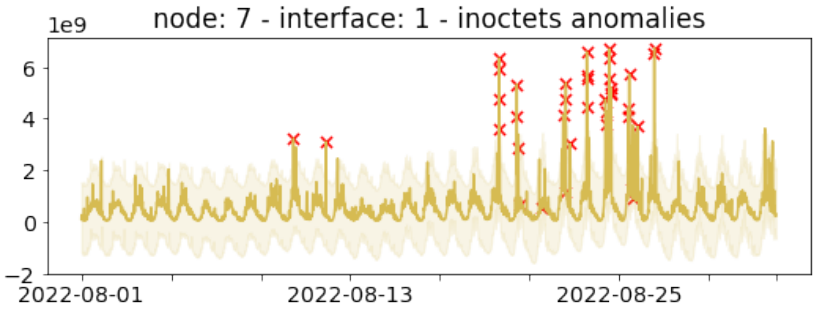
The reduced experimentation data set was standardized (see Section 3.2.1) using the `Pandas` library [39] [56] for `Python` before being used as input for the algorithms, to negate the effects of scale on their performance (standardization was chosen over normalization as a consequence of the presence of spike outliers in the data). Furthermore, `Pandas` was also used to impute missing data in the time series through linear interpolation (see Section 3.1.1), which was deemed a good compromise between precision and ease of implementation to fill in the gaps.

Because of the lack of an available labeled data set and the limited resources available, the algorithms from different libraries were compared based on visual inspection of their ability to identify data points that displayed seemingly anomalous behavior. Furthermore,

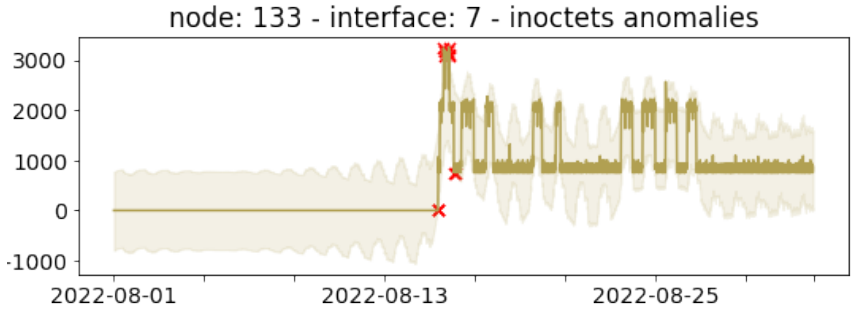
parameter tuning for these algorithms was carried out manually, as an objective measure of performance would be needed to automate the parameter selection process. A quantitative evaluation based on synthetic time series was later carried out for the models of the most promising library (see Section 4.6).

4.4.1. STL Residuals Detection

A first attempt at performing univariate anomaly detection was carried out by utilizing the results of the seasonal-trend decomposition described in Section 4.3. Each sample's residual component value is utilized as its anomaly score, and anomalies are identified as those samples whose score's distance from the data's mean exceeds N standard deviations (N=3 appears to offer the best results). Figure 4.5 displays an example of the results of this process, with the threshold for anomalies being indicated by the transparent confidence bounds, and anomalies being marked with red crosses. A more comprehensive view is provided in Appendix C.1.



(a) Time series with spikes.



(b) Time series with a changepoint and level shifts.

Figure 4.5: Examples of anomaly detection based on STL residuals (N=3). The x-axis represents time, while the y-axis represents the value of the time series. The red crosses in the plot are placed in correspondence of detected anomalies.

As expected, this method is particularly effective at identifying spike anomalies, but shows much less consistency in identifying changepoint and level shift outliers.

4.4.2. Quidditch Library

The first anomaly detection library used as part of the experimentation process was `Quidditch` [23], an inner-source framework developed internally at *Ericsson*. `Quidditch` offers implementations for many univariate and multivariate anomaly detection models, as well as functions for performing different time series processing operations, such as differencing, de-trending, and scaling. The library also contains the necessary infrastructure to set up anomaly detection pipelines and ensembles, as well as some tools to provide increased explainability of detection results. These factors make it a good candidate for the construction of the ADS.

From the set of `Quidditch` algorithms listed in Section 3.4.1, Isolation Forest and Random Cut Forest appeared to provide the most promising results. Nevertheless, after conducting various experiments, they still appeared to perform less optimally than the STL decomposition approach detailed in Section 4.4.1, being able to identify less apparent anomalies and instead mistakenly labeling a greater number of ordinary samples, while also requiring greater amounts of time to train.

Furthermore, experimenting with the `Quidditch` library brought to light several issues with its use as a basis for the ADS, despite the advantages brought by being an internal company tool:

- Some of the implemented algorithms contain bugs that make them difficult or impossible to use without modifying their code. This makes working with the library more cumbersome, but also poses doubts regarding the reliability of the library for use in production-grade systems.
- Because of the confidentiality of customer data, examples of projects making use of the library are difficult to access. Paired with its inner-source nature, this results in a severe lack of available example material on how to use the libraries tool effectively, which is available for most open-source frameworks.
- Since the library is an internal tool maintained by company employees, updates can be infrequent and dependent on the availability of dedicated personnel. As such, bugs may persist unaddressed for long stretches of time and new features and techniques may be slow to be added.

As a result of these considerations, `Quidditch` was put aside in the hopes of identifying a more suitable tool.

4.4.3. Kats Library

The second library used for experimentation was `Kats` [28], an open-source framework for time series analysis developed by *Facebook*'s Infrastructure Data Science team. `Kats` provides several outlier detection algorithms, each specialized in detecting a specific kind of anomaly. Furthermore, it offers various time series forecasting models, feature extraction modules, visualization tools, and parameter tuning support, making it a promising option for the development of the ADS.

Several of the algorithms offered by `Kats` and listed in Section 3.4.1 appeared to provide promising results upon visual inspection. In particular, the ability of some algorithms to identify specific kinds of anomalies could allow for a preliminary form of classification to be applied without the need for labels or additional data. Another advantage of the library is its support for parameter tuning. While the documentation available at the time of writing only provides examples of automating parameter selection for forecasting models, recent work by *Facebook*'s Infrastructure Data Science team indicates the possibility of using the tools provided by `Kats` to perform automatic model selection and parameter tuning of anomaly detection models without the need for a labeled data set. [16]

Despite these significant advantages, however, some drawbacks also emerged from the experiments conducted with the library:

- Some of the algorithms in the library present bugs that make their features unable to be used without altering the code base. Once again, this results in greater difficulty in using the library to its full potential, and does not inspire confidence regarding the employment of this tool in production systems.
- The documentation for the library is extremely variable in quality, with some features being documented extensively and others lacking any sort of explanation as to their usage and function. The example code provided by the authors for the library's tools also appears to be lagging behind some of its more recent additions, making it difficult for inexperienced users to familiarize themselves with it.
- Updates to the library's code appear to be infrequent, and issues opened on its repository's page have been left unaddressed for a significant amount of time (as of the writing of this thesis). This is not ideal for a library to be used in production systems, as any bugs or discovered vulnerabilities would have to be fixed manually instead of being able to rely on official updates.

As a result, despite its promise, `Kats` was deemed not ideal for use in the development of the ADS in its state at the time of writing, and experimentation was carried on with

other tools.

4.4.4. Merlion Library

The third and final anomaly detection library utilized for experimentation was `Merlion` [6], an open-source time series intelligence library developed and maintained by the *Salesforce* team. `Merlion` features implementations for a variety of popular time series forecasting and anomaly detection techniques, as well as support for various pre-processing and post-processing operations. The library also provides various tools for visualization, parameter tuning, evaluation, and ensembling, which would be useful in the development of an ADS. Furthermore, all anomaly detection and forecasting models in the library are constructed to provide a unified interface and inherit functionalities from some common base classes, ensuring that the framework can be easily expanded through the addition of new models. These factors make it ideal for the implementation of a resilient and expansible ADS.

All of the algorithms in the `Merlion` library allow post-processing rules to be defined, which influence how the anomaly scores obtained as outputs from the models are used to identify anomalies. Specifically, the library offers the possibility of employing an anomaly score calibrator to transform the anomaly scores so that they follow a standard normal distribution, which can be useful to ensure that the scores returned by different models may be directly compared. When using the calibrator, however, the accuracy of the algorithms appeared to worsen significantly, leading to the decision to ignore this feature when testing individual models. Furthermore, the library allows for the use of thresholds to sparsify the detected anomalies, by setting to zero those anomaly scores whose value does not surpass N standard deviations from the series' anomaly score mean. Experimentation lead to the decision of setting N=4 when not employing a calibrator, while if the latter is used a better choice appears to be N=3.

Several of the tested `Merlion` algorithms listed in Section 3.4.1 showed good performance upon visual inspection, and a more thorough evaluation of the models is provided in Section 4.6.2. Beyond the promising results obtained by its algorithms, the library also proved to be quite polished and robust during experimentation, along with possessing detailed documentation and being subjected to frequent updates. Furthermore, the tools provided by `Merlion` also include various ways to load and save model parameters, and to update trained models, which make the library a good candidate for use in a live system. Because of these considerations, it was decided to use `Merlion` as the building blocks for the ADS to be developed.

An example of anomaly detection conducted through the Prophet model, plotted using

the library's own tools, is shown in Figure 4.6. A more extensive display of the various models' predictions on the experimental data set is provided in Appendix C.2.

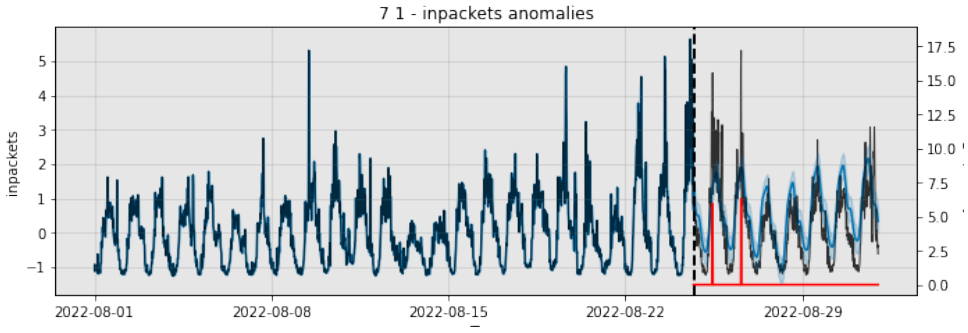


Figure 4.6: Anomaly detection using the Prophet model on a single time series feature. The x-axis represents time, while the y-axis reported on the left represents the value of the time series, and that on the right the value of anomaly scores. In the figure, the original time series is displayed in black, with a vertical dashed line dividing the training and test sets. The blue line indicates the model's forecast, as well as its confidence bound. The red vertical lines show instead the anomaly score of each sample in the test set, with the base being a score of zero.

In an attempt to further improve the accuracy of the algorithms, `Merlion`'s tools were also used to create ensembles of well-performing algorithms. These algorithms would each express an anomaly score for every data point, and their collective estimations would then be combined by taking either the mean, median, or maximum of the different models' scores. Upon visual inspection, the ensembles seemed to provide comparable results to those obtained by the individual best models, without a clear improvement or degradation of performance.

4.5. Model Evaluation Environment

As discussed in Section 3.5, the numerical evaluation of the models from the chosen library (`Merlion`) was carried out using a set of synthetically generated univariate time series. Because of this, the evaluation process could be carried out in a different computational environment with access to greater processing power, as there was no risk of leaking sensible customer data. The hardware used as part of this process is reported in Table 4.1.

RESOURCE	VALUE
CPU Model	8x 11th Gen Intel(R) Core(TM) i5-1145G7 @ 2.60 GHz
CPU Cache	8 MB
CPU Cores	4
RAM	24 GB
GPU Model	Mesa Intel(R) Xe Graphics (TGL GT2)
GPU Memory	3 GB

Table 4.1: Evaluation environment hardware specifications.

4.5.1. Synthetic Time Series Generation

The synthetic time series used for model evaluation were generated using the `GluonTS` library for `Python`. [2] As discussed in Section 3.5 this evaluation focuses on the detection of univariate anomalies, which made the process of generating time series considerably easier by restricting it to simulating a single variable at the time. The data was generated as a superposition of different numeric sequences with parameters reflecting those found in the `inoctets`, `outoctets`, `inpackets`, and `outpackets` features of the original data, with each value in the time series corresponding to one 15-minute interval sample. These sequences include:

- A smooth daily seasonality with sinusoidal behavior. This accounts for the main seasonality observed in the data.
- A series of additional seasonal sequences with shorter periods and lower, randomly-determined amplitudes. These account for other cyclic behavior observed in some of the data, such as the morning and afternoon having distinct peaks.
- A scaling factor differentiating between weekdays and weekends, and another differentiating between the days of the week.
- A sequence of Gaussian noise that simulates random fluctuations in the measured traffic.

The base time series is then further modified by injecting a random number of labeled behavioral anomalies and scaled to match the original data. The three types of behavioral anomalies described in Section 3.4.2 are added separately, and their labels are kept distinct to enable the computation of more detailed statistics for each category.

It is important to note that the anomaly label associated with level shift and changepoint

anomalies marks as anomalous only the moment of transition between the original and novel behavior, as, when the change persists for longer periods of time, no clear way was found to determine how long to consider the following observations as anomalous. The anomalies were simulated as follows:

- **Spike**: a sudden increase or drop of random intensity in the value of the time series, with a high chance of reverting to the original value with each new observation.
- **Level Shift**: a random shift in the scale and level of the time series, which reverts to the original after a random amount of time. With each change, the new level and scale have a small chance of becoming the normality, with subsequent shifts reverting to those conditions instead of the original ones.
- **Changepoint**: a random change between the original time series and a new sequence with different behavior. Each subsequent change has a small probability of returning the time series to its original behavior.

The likelihood of an anomaly occurring at a given sample is determined randomly. For spike anomalies, the probability of occurrence was set to an approximation of their frequency in the original data. Because level shifts and changepoints appear much more rarely, however, their chance of occurrence had to be increased so that they would appear in the evaluation data set in significant numbers. An example of generated time series is provided in Figure 4.7, with spike anomalies highlighted in red, level shift anomalies highlighted in blue, and changepoint anomalies highlighted in green.

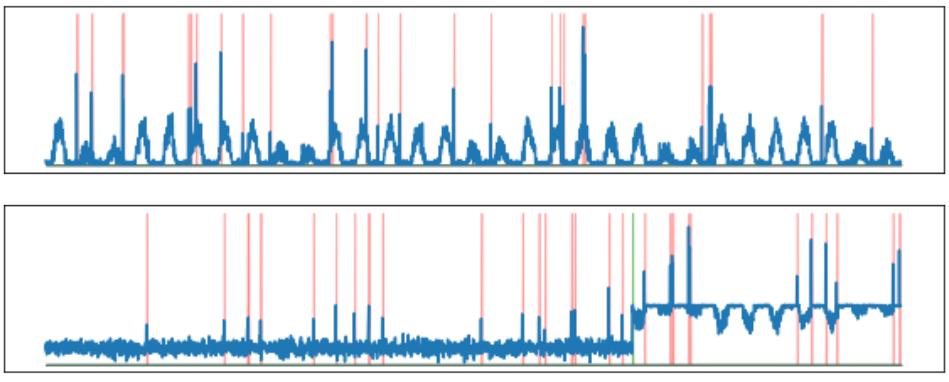


Figure 4.7: Examples of synthetic time series with injected anomalies. The x-axis represents time, while the y-axis represents the value of the time series.

4.6. Anomaly Detection Models Evaluation

Using `Merlion`'s evaluation tools, each model was tested on a set of 100 synthetic univariate time series of 45 days in length, with the first 31 days being used for the initial training of the model, and the remaining 14 to compute the evaluation metrics. The models were asked to predict the anomaly score of a single sample in the future at the time, thereby simulating an online detection environment and resulting in a total of 1344 predictions (96 per day). In order to limit the computational burden of the process, the models were retraining once every six hours of data on the samples from the most recent 31 days, though experimentation showed that more frequent re-trainings would lead to improved prediction accuracy. Each model was tested using the most promising set of hyperparameters identified during the experimentation phase, including general parameters such as pre-processing and post-processing steps, and model-specific parameters. In addition, the best-performing models were combined in an ensemble in an attempt to improve on their individual scores.

4.6.1. Model Hyperparameters

The general hyperparameters common to every evaluated model are reported in Table 4.2. Every model was tested after standardizing the data as described in Section 3.2.1, which was achieved through `Merlion`'s built-in pre-processing capabilities. Furthermore, each model's predicted anomaly scores were sparsified through the use of `Merlion`'s thresholding functions, setting any anomaly scores whose value failed to reach the cut-off point to zero. For this purpose, the threshold was set to a value of four standard deviations from the mean of anomaly scores for the given time series: this value was deemed optimal during the experimentation phase by visually inspecting the model's results on the reduced data set obtained from clustering. As described in Section 4.4.4, `Merlion` also provides a calibrator to transform anomaly scores to fit a standard normal distribution, but this feature was not utilized, as it showed a decrease in accuracy during the experimentation phase when applied to individual models.

The hyperparameters specific to each model are reported in Table 4.6.1. They are listed with the names and values used in the `Python` code, and any parameters not specified in the table were assigned the default value provided by the library.

MODEL	PARAMETER	VALUE
`DBL`	`train_window`	'4w'

MODEL	PARAMETER	VALUE
DBL	wind_sz	'7d'
	trends	['daily', 'weekly']
WindStats	wind_sz	60
	max_day	7
Spectral Residual	local_wind_sz	96*7
	q	5
Stat Treshold	-	-
ZMS	base	10
	lag_inflation	1.0
Isolation Forest	max_n_samples	None
	n_estimators	100
Random Cut Forest	n_estimators	100
	max_n_samples	8000
	online_updates	True
Autoencoder	hidden_size	5
	layer_sizes	(25, 10)
	sequence_len	1
	lr	10e-3
	batch_size	512
	num_epochs	20
VAE	encoder_hidden_sizes	(12, 6)
	decoder_hidden_sizes	(12, 6)
	latent_size	3
	sequence_len	1
	kld_weight	1.0

MODEL	PARAMETER	VALUE
VAE	dropout_rate	0.0
	num_eval_samples	10
	lr	10e-3
	batch_size	512
	num_epochs	10
DAGMM	gmm_k	3
	hidden_size	4
	sequence_len	1
	lambda_energy	0.1
	lambda_cov_diag	5*10e-3
	lr	10e-3
	batch_size	256
	num_epochs	10
LSTM ED	hidden_size	5
	sequence_len	1
	dropout	(0.3, 0.3)
	lr	10e-3
	batch_size	256
	num_epochs	30
	n_layers	(2,2)
Prophet	-	-

Table 4.3: Specific hyperparameters for the evaluated models.

PARAMETER	VALUE
Training set size	2976 samples (31 days)
Test set size	1344 samples (14 days)
Training frequency	24 samples (6 hours)
Detection frequency	1 sample (15 minutes)
Pre-processing	Input standardization
Post-processing	Anomaly score threshold (4σ)

Table 4.2: General hyperparameters for the evaluated models.

4.6.2. Model Evaluation Results

The results for each evaluated model on the chosen metrics (see Section 3.5) are reported in Table 4.4. The performance achieved by a `Random` model, which randomly assigns anomaly labels to each sample with 50% probability, is also reported at the top of the table for comparison. All performance metrics were computed as an average of those obtained by a given model on each of the 100 synthetic time series in the evaluation set (it should be noted that the average F1 score does not correspond to the F1 score calculated using the average precision and recall). For the Precision, Recall, and F1 Score metrics a higher value is desirable, while for Fallout, MTT, and MPT, lower values are preferred.

Furthermore, Table 4.5 reports the partial recall scores of each model for the three injected anomaly types, as described in Section 3.5.2.

From these results, the `Prophet` model was identified as achieving the best performance among those evaluated. The model surpasses all others both in precision and recall, resulting in a better overall F1 score, while retaining competitive fallout, and training time, although it shows a significant average prediction time when compared to most other options. The model also displays an outstanding performance when looking at specific types of anomalies, obtaining the best results on all three partial recall scores.

One aspect worth noting is the absence of model-specific hyperparameters for `Prophet`: while its performance surpassed all others during the course of this evaluation, it is nevertheless possible that a different choice of parameters for the other models might change this result. In particular, deep learning approaches like `Autoencoder`, `VAE`, `DAGMM`, and `LSTM ED` have a large number of parameters, and a more structured tuning approach making use of better hardware and more data could produce results far superior to those

Algorithm	Precision	Recall	Fallout	F1	MTT	MPT
Random	0.0081	0.5920	0.4999	0.0159	-	-
DBL	0.8246	0.6262	0.0014	0.6643	1.1563s	0.1862s
WindStats	0.5650	0.6666	0.0054	0.5691	0.0861s	0.0174s
Spec Residual	0.4538	0.14372	**0.0000**	0.1866	**0.0367s**	0.0135s
StatTreshold	0.8218	0.4824	0.0014	0.5402	0.0427s	**0.0124s**
ZMS	0.8721	0.4838	0.0005	0.5618	0.0579s	0.0234s
IsoForest	0.8543	0.5243	0.0004	0.5904	0.3681s	0.0910s
RanCutForest	0.3953	0.7903	0.0062	0.4704	2.4291s	1.0744s
Autoencoder	0.2419	0.4435	0.0138	0.2494	7.2149s	0.0546s
VAE	0.5457	0.7550	0.0087	0.5775	2.1497s	0.1092s
DAGMM	0.1613	0.3940	0.0040	0.1843	14.2508s	1.5305s
LSTM ED	0.4760	0.7556	0.0106	0.5228	11.9479s	0.1711s
Prophet	**0.8768**	**0.8116**	0.0008	**0.8231**	0.4706s	0.3673s

Table 4.4: Evaluation of the Merlion library models. The best overall results for each metric are highlighted in bold font.

achieved by those models during this study. Another element of note is that deep learning algorithms tend to benefit greatly from larger training sets, and thus using the same amount of data (31 days) for all models might disproportionately penalize these methods.

4.6.3. Training Data Evaluation Results

To better diagnose the different evaluated algorithms and identify possibilities for improvement, the models were also evaluated on their training data. This was achieved by using a similar process to that used for the testing data in Section 4.6.2, but instead including the testing data in the one used for training the model before computing the prediction. A comparison between the F1 and Fallout scores of the various models when tested on data used and not used in training is reported in Table 4.6.

In general, most models show a slight improvement in F1 score when tested on the training data, with the most significant discrepancy showing for the Random Cut Forest and DAGMM models, indicating perhaps an overfitting of the training data.

For models that base their prediction on comparing new observations with historical data, such as DBL, WindStats, StatThreshold, and ZMS, it appears that testing on the training

	Partial Recall		
Algorithm	**Spike**	**Level Shift**	**Changepoint**
Random	0.5914	0.6667	0.5454
DBL	0.6209	0.6111	0.7273
WindStats	0.6620	0.2222	0.7273
Spectral Residual	0.1394	0.1111	0.5454
Stat Treshold	0.4785	0.4444	0.7273
ZMS	0.4840	0.5	0.7273
Isolation Forest	0.5210	0.6667	0.8182
Random Cut Forest	0.7884	0.8334	0.6364
Autoencoder	0.4364	0.3333	0.8182
VAE	0.7517	0.4444	0.8182
DAGMM	0.3935	0.3889	0.3636
LSTM ED	0.7549	0.4444	0.6364
Prophet	**0.8066**	**0.8889**	**1.0000**

Table 4.5: Evaluation of the `Merlion` library models on specific types of anomalies. The best overall results for each metric are highlighted in bold font.

data can variably result in minor improvements or degradation of performance. This is likely because, when training on a new observation, if the observation is anomalous it results in a skew of the model's expectation of behavior away from the real normalcy of the time series, while training on ordinary data better aligns it to the truth. As such, depending on the number of anomalies contained in recent observations, training the model on the data it is then asked to analyze may yield both improved and worsened results.

For deep learning models, instead, the relatively small difference between the training and testing performance may suggest that a more accurate tuning of parameters may be required, in order to increase the size of the network and improve the models' ability to extrapolate the time series' behavior.

4.6.4. Post-processing Evaluation Results

An additional set of detection experiments was performed on the `Prophet` model to analyze its performance using different sets of post-processing options. This was carried out

	Training Data		New Data	
Algorithm	**Fallout**	**F1**	**Fallout**	**F1**
DBL	0.0011	**0.6649**	**0.0014**	0.6643
WindStats	**0.0051**	0.5383	0.0054	**0.5691**
Spec Residual	**0.0000**	**0.1927**	**0.0000**	0.1866
StatTreshold	**0.0013**	**0.5417**	0.0014	0.5402
ZMS	**0.0005**	0.5565	**0.0005**	**0.5618**
IsoForest	**0.0004**	0.5875	**0.0004**	**0.5904**
RanCutForest	0.0067	**0.5418**	**0.0062**	0.4704
Autoencoder	**0.0109**	0.2491	0.0138	**0.2494**
VAE	**0.0087**	**0.5775**	**0.0087**	**0.5775**
DAGMM	**0.0035**	**0.2265**	0.0040	0.1843
LSTM ED	**0.0112**	**0.5305**	0.0106	0.5228
Prophet	**0.0008**	**0.8305**	**0.0008**	0.8231

Table 4.6: Evaluation of the Merlion library models when tested on training vs. previously unseen data. The best results for F1 Score and Fallout of each model are highlighted in bold font.

to verify the conclusions drawn through visual inspection of the detection plots during the experimentation phase, to objectively determine the best combination of post-processing parameters to be applied to the best-performing model. The results of these experiments are reported in Table 4.7. The measures pertaining to computation times were omitted, as the effect on those metrics caused by the post-processing operations is negligible.

The results confirm that the use of the calibrator on the individual model results in a severe deterioration of its performance, even when using the threshold that results in the highest F1 score (which is proven to be three standard deviations). Without using anomaly score calibration, the threshold can nevertheless be tuned to increase Recall at the expense of Precision and Fallout or *vice versa*, but the value resulting in the best F1 score (while maintaining a low Fallout) is confirmed to be four standard deviations.

4.6.5. Ensemble Evaluation Results

In an attempt to improve the performances obtained from the evaluation of the individual models, the best-performing algorithms were merged in ensembles using Merlion's built-

`Prophet` Algorithm					
Calibration	**Threshold**	**Precision**	**Recall**	**Fallout**	**F1**
No	3σ	0.7882	**0.8757**	0.0013	0.7992
No	4σ	0.8641	0.8097	0.0011	**0.8121**
No	5σ	**0.8747**	0.7594	**0.0009**	0.7831
Yes	2σ	0.2287	0.3302	0.0107	0.1848
Yes	3σ	0.6164	0.2422	0.0011	0.3210
Yes	4σ	0.1870	0.0677	0.0004	0.0906

Table 4.7: Evaluation of the `Prophet` model using various post-processing parameters. The best overall results for each metric are highlighted in bold font.

in tools. The predictions of the ensembles are determined by combining the scores of the individual models through a specified policy, such as taking for each sample the mean of the models' predicted anomaly scores). It should be noted that because different models can produce anomaly scores with values at different scales, models assigning larger scores on average will have a greater influence on the ensemble's final prediction than the ones using smaller values. As a result, some ensembles were created and tested both with and without the use of the anomaly score calibration post-processing feature offered by `Merlion` (see Section 4.4.4), despite its tendency to decrease the model's performance when applied to individual models.

The evaluation of the ensemble models was once again performed on a smaller data set of synthetic time series because of the increased processing time required, and the results are reported in Table 4.8. The MTT and MPT scores of the models were omitted, as they roughly correspond to the sum of the individual models, with the process of merging the result being negligible in comparison. The results obtained by the best performing individual model (`Prophet`) on the smaller evaluation data set are reported twice at the top for comparison, both with a four standard deviations threshold and no calibration, and with a three standard deviation threshold alongside calibration.

In the table, the *Agg* column indicates the aggregation policy used to combine the individual models' anomaly scores. The *Post* column indicates instead the post-processing operation applied to the individual models. A *Post* value of C indicates the application of the anomaly score calibrator on each individual model, while a value of T indicates the use of a threshold on their score (set to three standard deviations when also using the calibrator, and four otherwise). A final threshold of four standard deviations (or three,

when using calibration) is always applied to the final aggregated anomaly score produced by the ensemble.

From these results it can be observed that some ensembles succeed in slightly surpassing the best individual model in terms of Recall or Fallout, but nevertheless fail to improve on the F1 score. These marginal improvements hardly justify the greatly increased processing time resulting from the use of multiple models simultaneously. As such, the individual `Prophet` model was determined to be the best suited to be used in the development of the final system.

Algorithms	Post	Agg	Precision	Recall	Fallout	F1
Prophet	T	-	**0.8641**	0.8097	0.0011	**0.8121**
	C+T	-	0.6164	0.2423	0.0012	0.3210
DBL+Prophet	-	Mean	0.8239	0.7922	0.0012	0.7751
	-	Max	0.8233	0.7932	0.0012	0.7752
	C	Mean	0.8391	0.7207	0.0010	0.7351
	C	Max	0.8164	0.7014	0.0010	0.7117
	T	Mean	0.8246	0.7963	0.0012	0.7765
	T	Max	0.8246	0.7911	0.0012	0.7736
	C+T	Mean	0.8176	0.7041	0.0010	0.7190
	C+T	Max	0.8179	0.7042	0.0011	0.7160
DBL+Prophet	C	Mean	0.8216	0.6266	0.0015	0.6688
+IsoForest	C	Max	0.7816	0.6001	**0.0005**	0.6435
	C	Median	0.7816	0.6035	**0.0005**	0.6457
	C	Mean	0.8319	**0.8289**	0.0016	0.7999
	C	Max	0.8627	0.7855	0.0012	0.7963
	C	Median	0.8636	0.7880	0.0013	0.7994
	T	Mean	0.8226	0.6178	0.0015	0.6591
	T	Max	0.7816	0.6009	**0.0005**	0.6430
	T	Median	0.7816	0.5858	**0.0005**	0.6339

Table 4.8: Evaluation of the ensemble models using various parameter combinations. The best overall results for each metric are highlighted in bold font. The *Agg* column indicates the policy for combining the models' anomaly scores. The *Post* column indicates the applied post-processing operation: C indicates the application of the anomaly score calibrator on each individual model, while T indicates the use of a threshold on their score.

4.7. Microservices Implementation

When referring to the solution architecture presented in Section 2.7, the components implemented during the course of this thesis are the data cleaning, model training, and anomaly detection services.

The *data cleaning service* performs the data cleaning operations described in Section 4.1 with the exception of step 3 (data anonymization), since in the context of live deployment on customer hardware there is no need to hide the information from the operator. As previously specified, each observation is associated with the network element to which it belongs and a timestamp indicating when it was sampled, as well as several features of interest to be analyzed. The cleaned data is then saved to the designated storage location, and the Model Training and Anomaly Detection services are informed of the presence of new data.

The *model training service* is tasked with creating and updating the models used to perform anomaly detection according to some specified parameters. For the univariate detection algorithms, a new model has to be created for each feature of every network interface, whereas for multivariate detection algorithms, a single model can be generated for each interface. When the service cannot identify an existing model for a given feature or interface in the designated storage location, it will generate a new one according to the specified parameters (including which kind of model to choose, what parameters to use, and what pre-processing and post-processing operations should be performed). When a model already exists, the system will instead calculate the time elapsed between the last sample used in training and the most recent available sample in the data for which anomaly detection has already been performed. If this quantity exceeds the amount specified in the service parameters, the model will be updated by training it on the more recent data cleaned by the Data Cleaning service.

The *anomaly detection service*, finally, provides an estimation of anomaly scores for each data point that has not yet been analyzed. By default, the system performs its prediction on the data as soon as a new sample is available, but the service parameters may be changed so that the detection process is executed with a specific cadence. The detection process is composed of the following steps:

1) The data is loaded from the designated storage location. The amount of data to be loaded is determined by the service configurations, and includes both the most recent observations (for which detection has to be performed), as well as a specified amount of samples preceding them (necessary for the use of some models).

2) The system identifies missing samples, which are marked by the presence of `NaN` values in the data features, and flags them as anomalies.

3) The system analyzes the data's features, flagging those values that fall outside a predefined domain range that can be specified for each feature (which defaults to accepting any value). For example, all of the features used as part of this study (listed in Section 4.1) correspond to the counting of a certain number of packets or bytes in the network, and as such, negative values should be always be flagged as errors. Furthermore, five of the features are themselves indicators for the presence of errors or discarded packets in the network, and should thus be flagged as anomalous whenever their value exceeds zero.

4) The system imputes the values according to the specified imputation policy, and applies a given scaling technique to the data. By default, these are set to linear interpolation and standardization respectively.

4) The system loads the appropriate anomaly detection model for the given network interface. Only the subset of specified features of interest is analyzed in this phase. If the service is configured to perform univariate detection it will load a model for each of its features, otherwise a single multivariate model will be used. The model flags each data point that it deems an outlier, and provides an anomaly score to indicate the magnitude of the anomaly (univariate models will each provide a score for their designated feature, while multivariate models will return a single value for each observation).

5) The anomaly labels and scores calculated during the previous steps are aggregated and stored in the designated storage location.

The implemented system is configured to utilize by default the `Prophet` model for univariate detection, and the `Isolation Forest` model for multivariate detection, with univariate detection being the default choice. Nevertheless, the ADS is structured in such a way that any other models from the `Merlion` library may be easily be used instead by modifying the appropriate configurations.

The output of the system is a new data point for each analyzed observation, which bears the latter's node ID, interface IDs, and timestamp. This data point, however, does not possess the same features as the original observation, but is instead associated with several new features indicating the presence or the absence of various kinds of anomalies.

The first set of features (identified by the suffix `_missing_anom`), indicates with a boolean value whether the specified feature was missing from the sample. In that case, no other

anomalies will be reported for that feature in the same sample.

The second set of features (identified by the suffix `(_value_anom)`), indicates with a boolean value whether the specified feature assumed a value outside its established domain in the given sample. In that case, no behavioral anomalies will be reported for that feature in the same sample.

The third and final set of features(identified by the suffix `_behavior_anom`), indicates the anomaly score associated with the given sample, as provided by the detection model. This analysis is only executed for a specified subset of features, and a non-zero score marks the feature's value as anomalous.

An example of the schema of the system's output for a data point in the experimentation data set is provided in Figure 4.8.

```
RangeIndex: 1 entries, 0 to 0
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   node                            1 non-null      object
 1   interface                       1 non-null      object
 2   timestamp                       1 non-null      datetime64[ns]
 3   inoctets_missing_anom           1 non-null      bool
 4   outoctets_missing_anom          1 non-null      bool
 5   inpackets_missing_anom          1 non-null      bool
 6   outpackets_missing_anom         1 non-null      bool
 7   ifInDiscardPkts_missing_anom    1 non-null      bool
 8   ifOutDiscardPkts_missing_anom   1 non-null      bool
 9   ifInErrorPkts_missing_anom      1 non-null      bool
 10  ifOutErrorPkts_missing_anom     1 non-null      bool
 11  CrcErrorPkts_missing_anom       1 non-null      bool
 12  inoctets_value_anom             1 non-null      bool
 13  outoctets_value_anom            1 non-null      bool
 14  inpackets_value_anom            1 non-null      bool
 15  outpackets_value_anom           1 non-null      bool
 16  ifInDiscardPkts_value_anom      1 non-null      bool
 17  ifOutDiscardPkts_value_anom     1 non-null      bool
 18  ifInErrorPkts_value_anom        1 non-null      bool
 19  ifOutErrorPkts_value_anom       1 non-null      bool
 20  CrcErrorPkts_value_anom         1 non-null      bool
 21  inoctets_behavior_anom          1 non-null      float64
 22  outoctets_behavior_anom         1 non-null      float64
 23  inpackets_behavior_anom         1 non-null      float64
 24  outpackets_behavior_anom        1 non-null      float64
dtypes: bool(18), datetime64[ns](1), float64(4), object(2)
```

Figure 4.8: Example schema of the anomaly detection service's output, as provided by `Pandas`.

5 | Conclusions

The following chapter draws from the contents of Chapter 4 to discuss the results of this thesis in relation to the goals established in Chapters 1 and 2. Section 5.1 summarizes the contents of this thesis, and the answer it provides to its driving research question. Section 5.2 analyzes the success of the system in respecting the specifications that guided its development. Section 5.3, finally, lists a series of potential improvements over the contents of this thesis that could be carried out in future work.

5.1. Thesis Summary

The purpose of this thesis was to develop an autonomous anomaly detection system, to be deployed as part of a software-defined networking controller platform developed at *Ericsson*.

The purpose of this thesis was to answer the research question described in Section 1.1.1: 'How could an anomaly detection system for reduced amounts of mobile network operational data be constructed?'. The goal of this system was to unsupervisedly detect anomalous behavior occurring in a number of monitored network metrics, in order to provide diagnostic support to network operators and enable the creation of automated responses to the presence of abnormalities. To ensure that the system would be adaptable and scalable to networks of changing characteristics and sizes, the system was constructed using machine learning technology and in accordance with microservices architecture.

The data used in this study covered six months of regularly sampled multivariate time series data from a real-life network. This data was cleaned and processed using `PySpark` to enable its use by autonomous anomaly detection algorithms.

Experimentation of such algorithms was carried out on a reduced data set of time series with varied behavior, obtained by performing clustering on the original data and taking one time series from each group. Several attempts at clustering were made, with the best results being obtained by using the K-Means algorithm with dynamic time warping as a similarity measure.

Several anomaly detection techniques were tested on the data, drawing from the tools available across three different anomaly detection libraries for `Python`. In the end, the `Merlion` open-source library was chosen to be used in the development of the anomaly detection models employed by the system, as a result of the wide range of algorithms and tools it provides, its extensive documentation, and its reliable code base.

The library's algorithms were evaluated across a variety of metrics on a data set of synthetic time series, generated appositely to resemble the original data. The models' accuracy and computational requirements were compared on a univariate detection problem, with the `Prophet` model emerging as the most promising among those tested.

In answer to the research question of this thesis, the anomaly detection system was implemented as a set of three microservices, each carrying out data cleaning, model training, and anomaly detection activities respectively. It is constructed using scalable and reliable technologies, which ensure it is adaptable to different network specifics, and is configurable to enable finer control over its performance trade-offs. Its anomaly detection service is capable of identifying different types of faults in the data, including behaviors that diverge from the system's historical patterns, which it achieves through a mix of static and machine learning-based techniques. The system was built to be highly customizable by the end user to fit their specific needs, and to be easily maintainable and expansible by the *Ericsson* team through future work.

5.2. Satisfaction of System Specifications

The anomaly detection system developed throughout the course of this thesis was implemented according to the specifications described in Section 2.7.1. With regard to those specifications, the following conclusions can be drawn from the final result:

S1) The system should identify the majority of anomalies occurring within the data:
The developed ADS is capable of detecting missing data, as well as samples whose values exceed or fall below set domain limits (see Section 4.7). Furthermore, the system uses autonomously trained, manually configurable machine learning models to determine unusual behavior within the data. Its models are estimated to be capable of detecting an average of 81.16% of anomalies occurring within a given monitored metric, if trained every six hours (see Section 4.6.2).

S2) The system should not mark a large number of ordinary observations as anomalous:

The developed ADS is estimated to perform erroneous detections on an average of only 0.08% of the analyzed data, when its models are trained every six hours (see Section 4.6.2). Additional parameters may be configured by its operators to reduce this quantity, such as by raising the threshold used to determine anomalous occurrences.

S3) The system should be able to adapt to the appearance of novel behaviors in the monitored metrics:
By choosing to employ lightweight and highly specialized models, the system is capable of retraining itself to adapt to new behavioral patterns with any desired frequency (at the cost of increased computational burden). Furthermore, because these models require only a small amount of data to be trained and can be deployed for a single time series, the system can quickly adapt to the addition of new monitoring metrics.

S4) The system should provide online predictions within the span of one sampling interval:
The developed ADS employs models that are capable of expressing predictions on a given metric within an estimated average of 0.3673 seconds (see Section 4.6.2). While during this thesis no estimation was calculated on the time required for the execution of the remaining parts of the system's functions, its components were developed as individual microservices using scalable technologies (see Section 4.7), which should ensure that the system can always reach a sufficient performance when provided with the necessary processing power.

S5) The system should be able to function with low amounts of processing power:
The developed ADS was implemented favoring modern and efficient technologies. Furthermore, the system allows for a significant degree of customization, which makes it possible to sacrifice its accuracy in exchange for a reduced computational burden, for example by using a lighter detection model or increasing the time between re-trainings (see Section 4.7).

S6) The system should be able to scale to handle varying amounts of input data:
The developed ADS was implemented using scalable technologies and frameworks, such as `PySpark` and MSA (see Sections 4.1 and 4.7). As a result, the system can be efficiently scaled to adapt to IP networks of smaller or larger sizes.

S7) The system should be built with reliable, stable technologies: The de-

veloped ADS was built using open-source, well-supported libraries, such `PySpark`, `Pandas`, and `Merlion` (see Sections 4.1, 4.4, and 4.7). These libraries were deemed to be both stable and reliable.

S8) The system should be easy to maintain and expand upon in future work: The developed ADS was built with the specific intention of being expansible through future developments (see Section 2.7). Furthermore, the adhesion to MSA ensures that the different parts of the system can easily be modified without impacting the rest of the system (see Section 4.7). Finally, the system was built using well-documented, open-source libraries, and its code was documented painstakingly throughout the course of this work.

It can be concluded that the system broadly meets the defined specifications, and can successfully accomplish its stated goal of performing autonomous anomaly detection on a provided set of arbitrary network metrics.

5.3. Future Development

Numerous improvements could be carried out in future work to improve or expand on the contents of this thesis.

Firstly, as described in Section 2.7, the system should be expanded by the addition of an *anomaly classification service* and a *rule engine service*. The former would perform classification on the detected anomalies to divide them into different groups. A preliminary version of this process could be carried out by using algorithms capable of detecting specific types of anomalies, but a more advanced approach, possibly incorporating additional data to be requested from the IP network, should also be taken into consideration. The latter service would instead provide a set of configurable rules to the network operator, which would allow them to customize the alerts provided by the system, as well as set up automated responses to detected anomalies (such as requesting additional, detailed data), or to the output of the classification process.

In addition to this, the amount and variety of available data should be improved. An example would entail using observations over longer periods of time, which would enable analysis of anomalies occurring over the network's quarterly and yearly behavior, and better take into account the effects of holidays and other yearly trends. Data could also be drawn from different networks, to enable a better view of the time series' behavior over different geographies and uses. Another option would entail collecting data with a higher sampling frequency to analyze finer changes in the monitored metrics, and additional

metrics such as error logs could also be analyzed using text analysis techniques. More importantly, data containing labeled anomalies should be obtained, to gain better insight into which anomalous behaviors in the monitored metrics actually correspond to real-world events of interest.

The anomaly detection process itself could be improved in multiple ways. To begin with, the algorithms tested as part of this thesis should be evaluated in greater detail, for example by performing more accurate hyperparameter tuning and training more powerful models with larger amounts of data. More detection models could also be developed to work in support of the existing system, such as by detecting anomalies at different time scales, or by performing multivariate anomaly detection alongside the univariate one. Additional experiments could then be conducted by testing a wider array of detection algorithms and libraries.

A more extensive evaluation of the system's global performance should also be undertaken. This includes testing the performance of its components other than anomaly detection (such as the data cleaning pipeline), but also verifying its ability to scale when faced with varying loads of input data. Beyond this, a more rigorous model and data versioning system could be integrated with the existing infrastructure, along with automated forms of testing and deployment in accordance with MLOps best practices from within the industry.

The clustering process described in this thesis work could also be improved. During this work, clustering was only carried out to obtain a varied data set to be used in experimentation, and thus a limited portion of the available time was spent in its implementation. Potential improvements would include experimentation with different approaches, such as by combining DTW with algorithms other than K-Means, and a more rigorous selection of optimal parameters than the use of the loathed elbow method, but also simply utilizing more powerful hardware and a greater amount of data.

Another improvement could take the form of developing cluster-specific models, as discussed in Section 3.4. This would entail the use of clustering on the live data coming into the system, and would have the double advantage of allowing for a lower number of models to be created and trained (one for each cluster, rather than one for each network interface metric), while also making it possible for models to be trained on a wider range of time series (enabling the use of more powerful, generalizing models). An attempt could also be made to develop one single, powerful, general model capable of detecting anomalies on any time series in the data, given a sufficiently large and varied training set.

Finally, a vast improvement to the system would be the addition of an AutoML system.

Such a system would be capable of tuning and selecting different models based on their performance on the data. In order to do this, a labeled data set would be required, which is difficult to obtain for anomaly detection tasks under normal circumstances, but could be approximated by automating the process of synthetic time series generation. One such approach has been proposed by the *Facebook* Data Science Team [16] by applying seasonal decomposition on the input data, removing the contributions of the residual component, and adding randomized anomalies to the resulting time series. Another option would instead be to develop a generative model capable of generating anomaly-free time series similar in behavior to those sampled by the network, on which anomalies would once again be injected randomly.

Bibliography

[1] C. C. Aggarwal. *Outlier Analysis*. Springer, 2013. ISBN 978-1-4614-6395-5. doi: 10.1007/978-1-4614-6396-2. URL https://doi.org/10.1007/978-1-4614-6396-2.

[2] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang. Gluonts: Probabilistic time series models in python, 2019.

[3] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR. URL https://proceedings.mlr.press/v27/baldi12a.html.

[4] T. Benson, A. Akella, and D. A. Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[5] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and Communication Networks*, 9(18):5803–5833, 2016. doi: https://doi.org/10.1002/sec.1737. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737.

[6] A. Bhatnagar, P. Kassianik, C. Liu, T. Lan, W. Yang, R. Cassius, D. Sahoo, D. Arpit, S. Subramanian, G. Woo, A. Saha, A. K. Jagota, G. Gopalakrishnan, M. Singh, K. C. Krithika, S. Maddineni, D. Cho, B. Zong, Y. Zhou, C. Xiong, S. Savarese, S. Hoi, and H. Wang. Merlion: A machine learning library for time series, 2021.

[7] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano. A review on outlier/anomaly detection in time series data. *ACM Comput. Surv.*, 54(3), apr 2021. ISSN 0360-0300. doi: 10.1145/3444690. URL https://doi-org.libproxy.aalto.fi/10.1145/3444690.

[8] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Anomaly detection using autoencoders in high performance computing systems. *Proceed-*

ings of the AAAI Conference on Artificial Intelligence, 33(01):9428–9433, Jul. 2019. doi: 10.1609/aaai.v33i01.33019428. URL https://ojs.aaai.org/index.php/AAAI/article/view/4993.

[9] M. Braei and S. Wagner. Anomaly detection in univariate time-series: A survey on the state-of-the-art, 2020.

[10] E. Burnaev and V. Ishimtsev. Conformalized density- and distance-based anomaly detection in time-series data, 2016.

[11] R. J. G. B. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In J. Pei, V. S. Tseng, L. Cao, H. Motoda, and G. Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37456-2.

[12] S. Chakraborty, S. Shah, K. Soltani, A. Swigart, L. Yang, and K. Buckingham. Building an automated and self-aware anomaly detection system, 2020.

[13] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3), jul 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL https://doi-org.libproxy.aalto.fi/10.1145/1541880.1541882.

[14] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012. doi: 10.1109/TKDE.2010.235.

[15] C. Chatfield. *Time-series forecasting*. CRC press, 2000.

[16] S. Chatterjee, R. Bopardikar, M. Guerard, U. Thakore, and X. Jiang. Mospat: Automl based model selection and parameter tuning for time series anomaly detection, 2022.

[17] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition. *J. Off. Stat*, 6(1):3–73, 1990.

[18] D. Comer and A. Rastegarnia. Toward disaggregating the sdn control plane. *IEEE Communications Magazine*, 57(10):70–75, 2019. doi: 10.1109/MCOM.001.1900063.

[19] A. A. Cook, G. Mısırlı, and Z. Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2019.

[20] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer

International Publishing, Cham, 2017. ISBN 978-3-319-67425-4. doi: 10.1007/978-3-319-67425-4_12. URL https://doi.org/10.1007/978-3-319-67425-4_12.

[21] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[22] P. A. Gagniuc. *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[23] E. GAIA. Quidditch: A framework for solving anomaly detection problems. Gitlab, 2020. URL https://gitlab.internal.ericsson.com/gaia/projects/anomaly-detection/quidditch.

[24] S. Guha, N. Mishra, G. Roy, and O. Schrijvers. Robust random cut forest based anomaly detection on streams. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2712–2721, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/guha16.html.

[25] D. Hawkins. *Identification of outliers*. Monographs on applied probability and statistics. Chapman and Hall, London [u.a.], 1980. ISBN 041221900X. URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+02435757X&sourceid=fbw_bibsonomy.

[26] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. V. Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, M. Dumonal, J. Kooscisz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch. Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, 23(124):1–6, 2022. URL http://jmlr.org/papers/v23/21-1177.html.

[27] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 387–395, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219845. URL https://doi-org.libproxy.aalto.fi/10.1145/3219819.3219845.

[28] X. Jiang, S. Srivastava, S. Chatterjee, Y. Yu, J. Handler, P. Zhang, R. Bopardikar, D. Li, Y. Lin, U. Thakore, M. Brundage, G. Holt, C. Komurlu, R. Nagalla, Z. Wang, H. Sun, P. Gao, W. Cheung, J. Gao, Q. Wang, M. Guerard,

M. Kazemi, Y. Chen, C. Zhou, S. Lee, N. Laptev, T. Levendovszky, J. Taylor, H. Qian, J. Zhang, A. Shoydokova, T. Singh, C. Zhu, Z. Baz, C. Bergmeir, D. Yu, A. Koylan, K. Jiang, P. Temiyasathit, and E. Yurtbay. Kats, 3 2022. URL `https://github.com/facebookresearch/Kats`.

[29] P. Jinka and B. Schwartz. *Anomaly detection for monitoring: A statistical approach to time series anomaly detection*. O'Reilly Media, 2015.

[30] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013. doi: 10.1109/MCOM.2013.6461195.

[31] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[32] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[33] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103:14–76, 1 2015. ISSN 15582256. doi: 10.1109/JPROC.2014.2371999.

[34] K.-H. Lai, D. Zha, G. Wang, J. Xu, Y. Zhao, D. Kumar, Y. Chen, P. Zumkhawaka, M. Wan, D. Martinez, and X. Hu. Tods: An automated time series outlier detection system. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):16060–16062, May 2021.

[35] J. R. G. Lansangan and E. B. Barrios. Principal components analysis of nonstationary time series data. *Statistics and Computing*, 19:173–187, 6 2009. ISSN 09603174. doi: 10.1007/S11222-008-9082-Y/METRICS. URL `https://link-springer-com.libproxy.aalto.fi/article/10.1007/s11222-008-9082-y`.

[36] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi: 10.1109/ICDM.2008.17.

[37] J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967.

[38] M. Müller. *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg, Berlin,

Heidelberg, 2007. ISBN 978-3-540-74048-3. doi: 10.1007/978-3-540-74048-3_4. URL https://doi.org/10.1007/978-3-540-74048-3_4.

[39] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL https://doi.org/10.5281/zenodo.3509134.

[40] K. Pearson. LIII. On lines and planes of closest fit to systems of points in space, Nov. 1901. URL https://doi.org/10.1080/14786440109462720.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[42] F. Petitjean, A. Ketterlin, and P. Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693, 2011. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2010.09.013. URL https://www.sciencedirect.com/science/article/pii/S003132031000453X.

[43] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh. Dynamic time warping averaging of time series allows faster and more accurate classification. In *2014 IEEE International Conference on Data Mining*, pages 470–479, 2014. doi: 10.1109/ICDM.2014.27.

[44] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM workshop on hot topics in networks*, pages 43–48, 2012.

[45] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, jul 2019. doi: 10.1145/3292500.3330680.

[46] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL https://www.sciencedirect.com/science/article/pii/0377042787901257.

[47] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978. doi: 10.1109/TASSP.1978.1163055.

[48] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13 (7):1443–1471, 2001.

[49] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[50] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020. URL `http://jmlr.org/papers/v21/20-091.html`.

[51] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72 (1):37–45, 2018. doi: 10.1080/00031305.2017.1380080. URL `https://doi.org/10.1080/00031305.2017.1380080`.

[52] P. W. Tsai, C. W. Tsai, C. W. Hsu, and C. S. Yang. Network monitoring in software-defined networking: A review. *IEEE Systems Journal*, 12:3958–3969, 12 2018. ISSN 19379234. doi: 10.1109/JSYST.2018.2798060.

[53] G. van Rossum. Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.

[54] Y. Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American statistical association*, 91(433):365–377, 1996.

[55] T. Warren Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2005.01.025. URL `https://www.sciencedirect.com/science/article/pii/S0031320305001305`.

[56] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[57] J. A. Wickboldt, W. P. De Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville. Software-defined networking: management requirements and challenges. *IEEE Communications Magazine*, 53(1):278–285, 2015.

[58] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu. A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys 'I&' Tutorials*, 21(1):393–430, 2019. doi: 10.1109/COMST.2018.2866942.

[59] G. Xu, Y. Mu, and J. Liu. Inclusion of artificial intelligence in communication networks and services. *ITU J. ICT Discov. Spec*, 1:1–6, 2017.

[60] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 187–196, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3185996. URL `https://doi-org.libproxy.aalto.fi/10.1145/3178876.3185996`.

[61] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[62] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJJLHbb0-`.

A | First Appendix - Clustering

This appendix displays the results obtained from the execution of the various clustering approaches detailed in Section 4.2. Each figure reports the results of one clustering attempt, with every plot being the result of the semi-transparent superimposition of all the time series in one cluster (in grey), along with one red time series calculated as the average of the group's members using DBA.

A.1. K-Means Clustering with DTW

A.1.1. Normalized - 20 Clusters

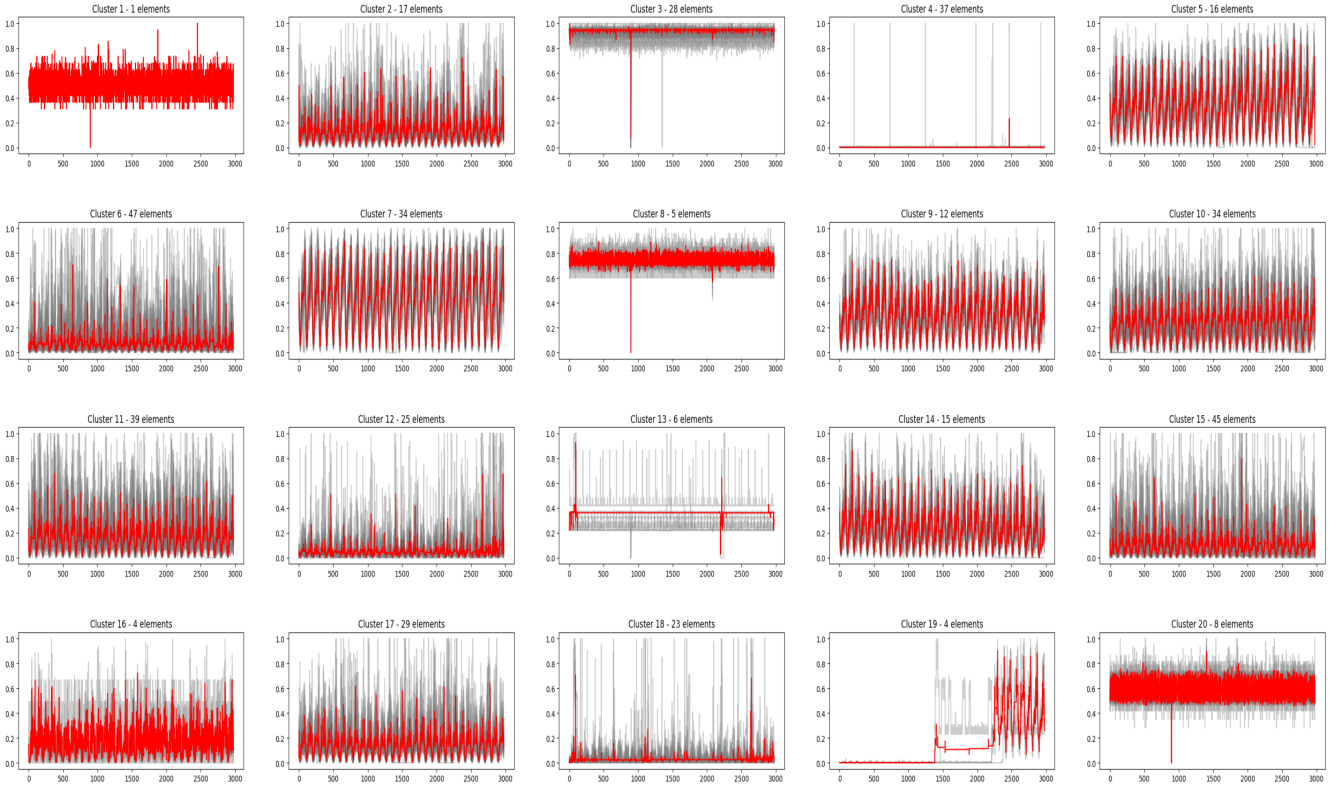


Figure A.1: Clustering of normalized time series in 20 clusters using K-Means with DTW.

A.1.2. Standardized - 20 Clusters

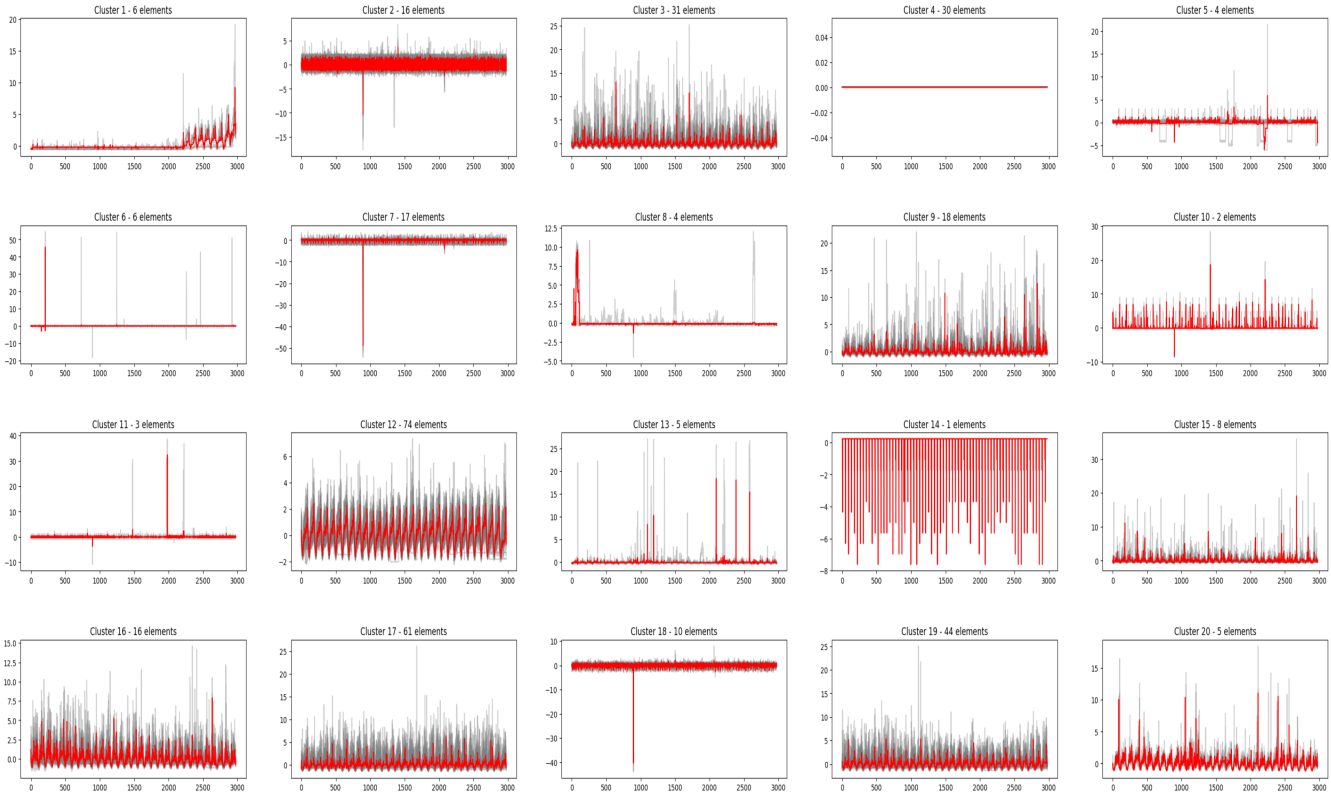


Figure A.2: Clustering of standardized time series in 20 clusters using K-Means with DTW.

A.1.3. Standardized - 30 Clusters

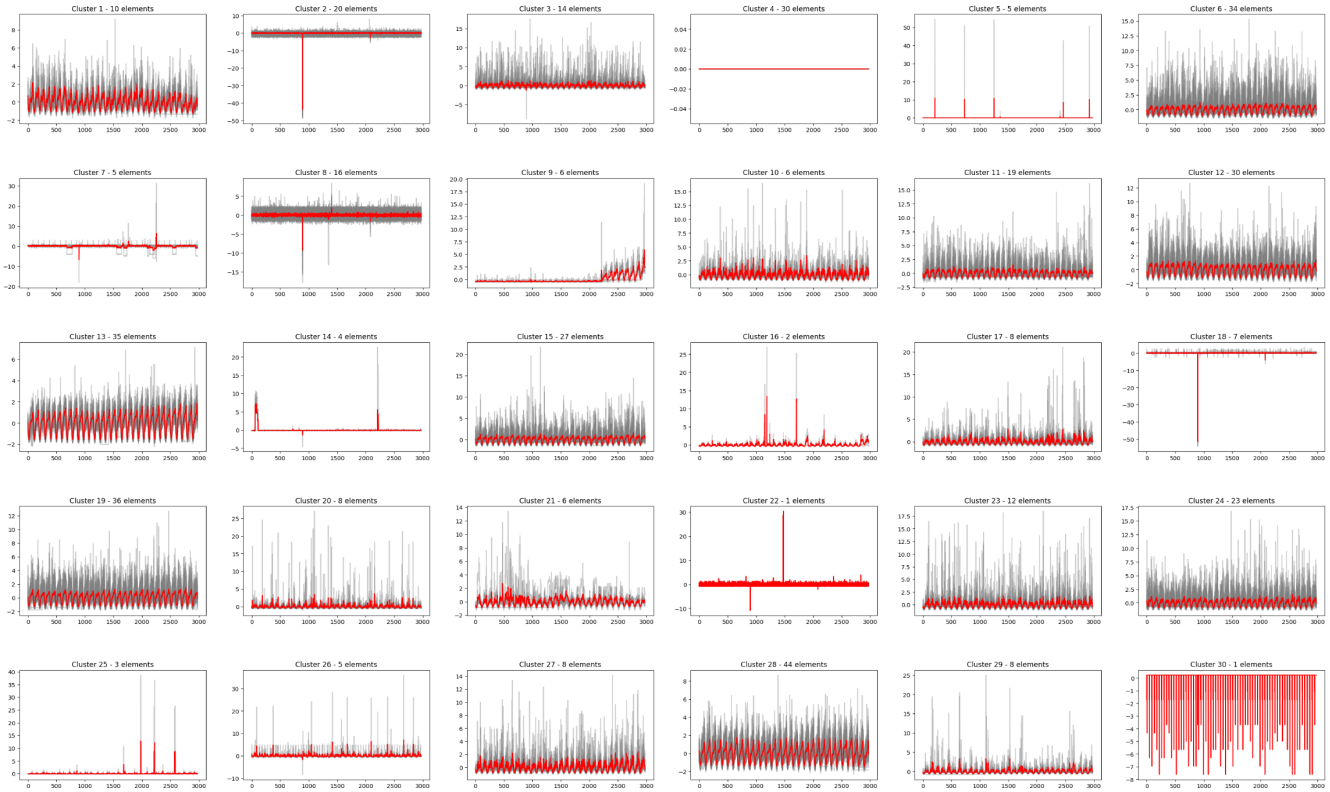


Figure A.3: Clustering of standardized time series in 30 clusters using K-Means with DTW.

A.2. K-Means Clustering with PCA

A.2.1. Normalized - 2 Principal Components - 27 Clusters

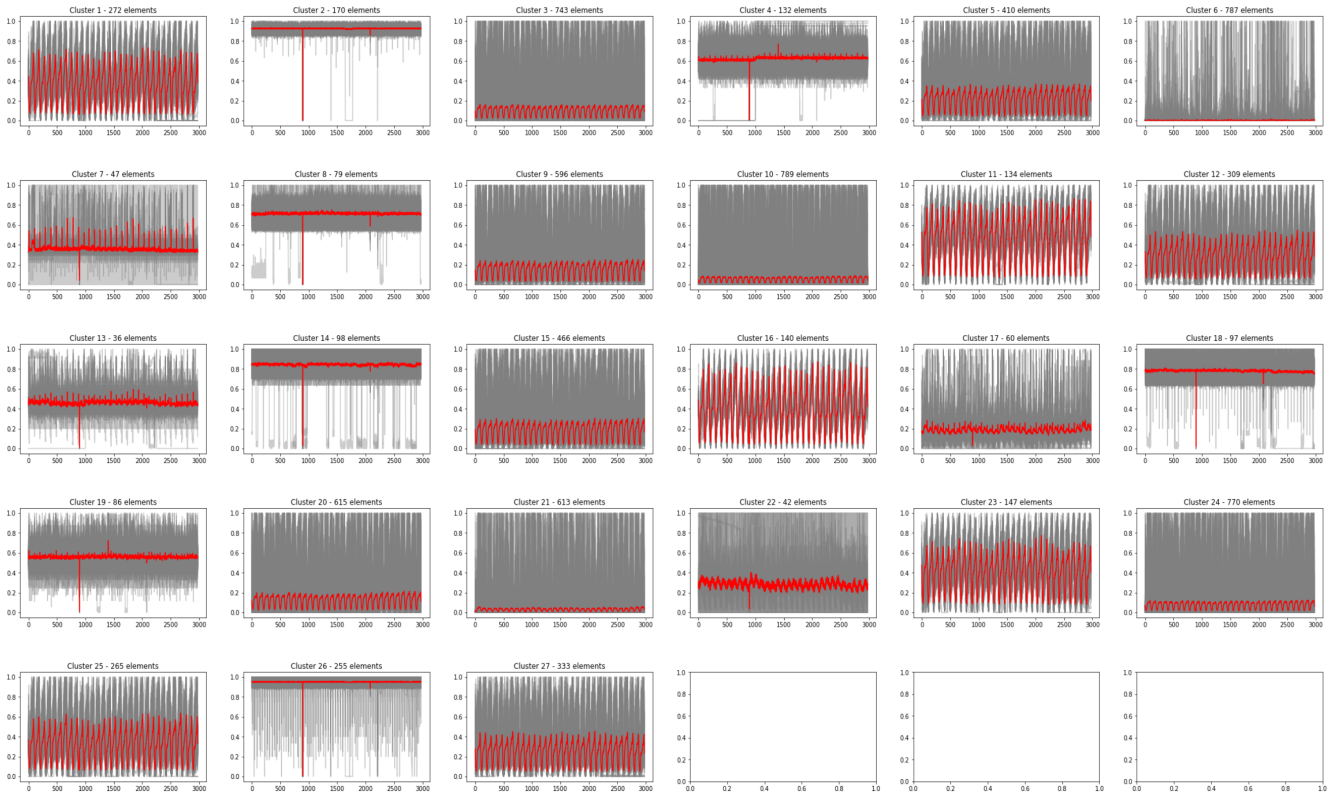


Figure A.4: Clustering using K-Means of normalized time series, transformed using PCA with 2 principal components, in 27 clusters.

A.2.2. Normalized - 64 Principal Components - 26 Clusters

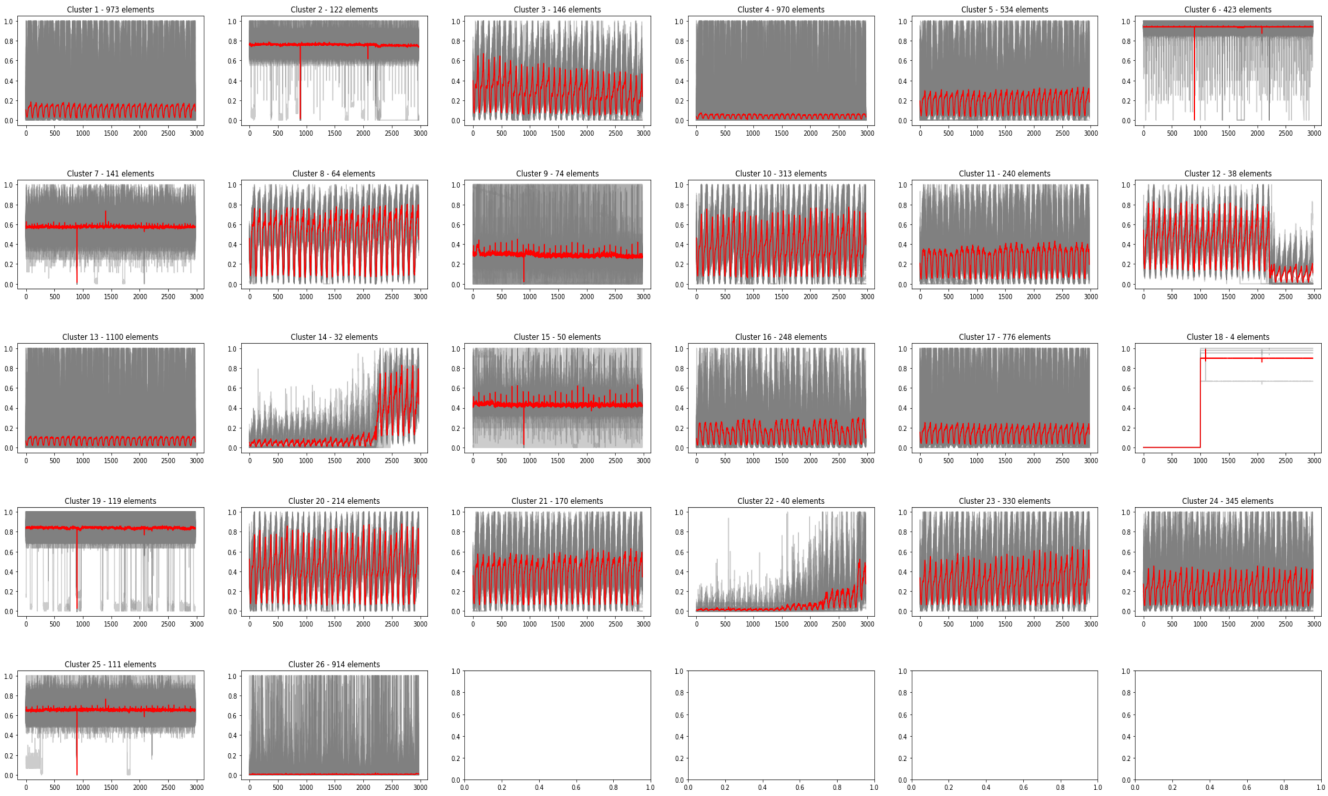


Figure A.5: Clustering using K-Means of normalized time series, transformed using PCA with 64 principal components, in 26 clusters.

A.2.3. Standardized - 2 Principal Components - 20 Clusters

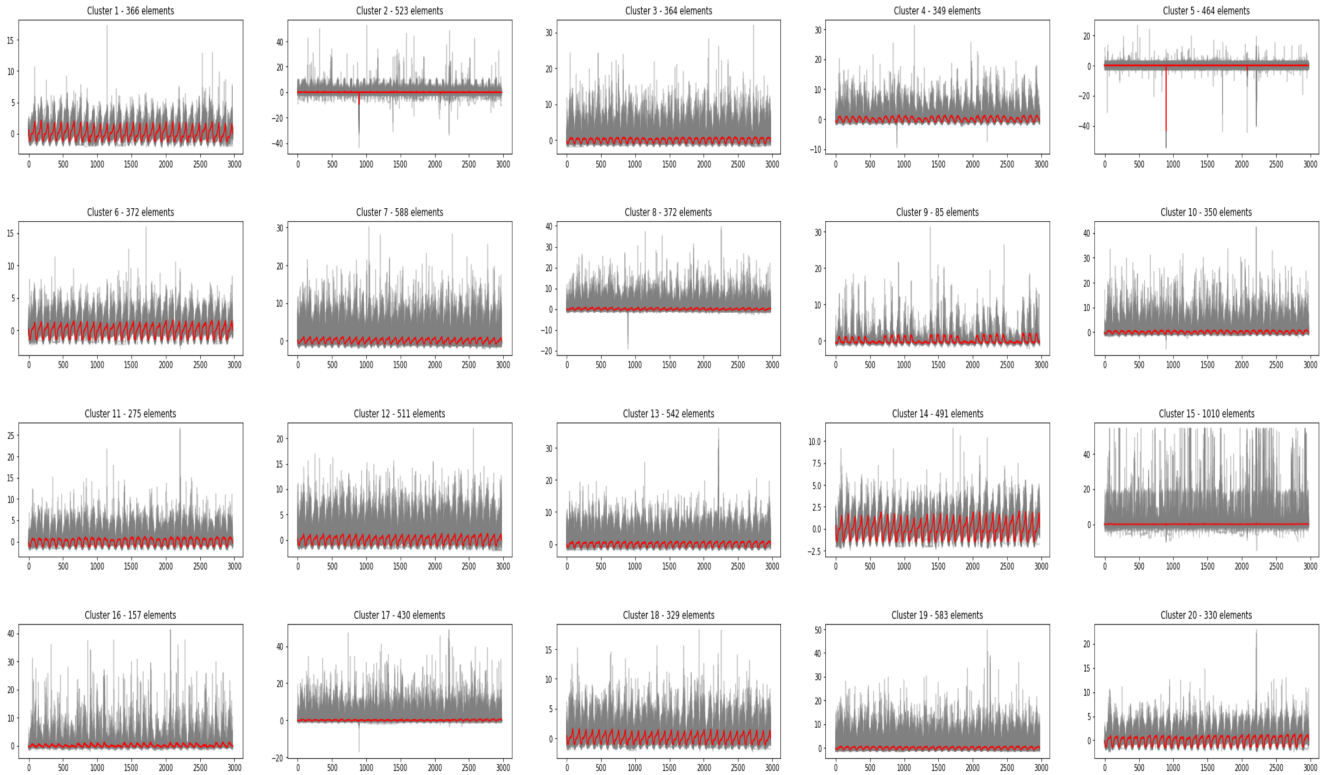


Figure A.6: Clustering using K-Means of standardized time series, transformed using PCA with 2 principal components, in 20 clusters.

A.2.4. Standardized - 120 Principal Components - 24 Clusters

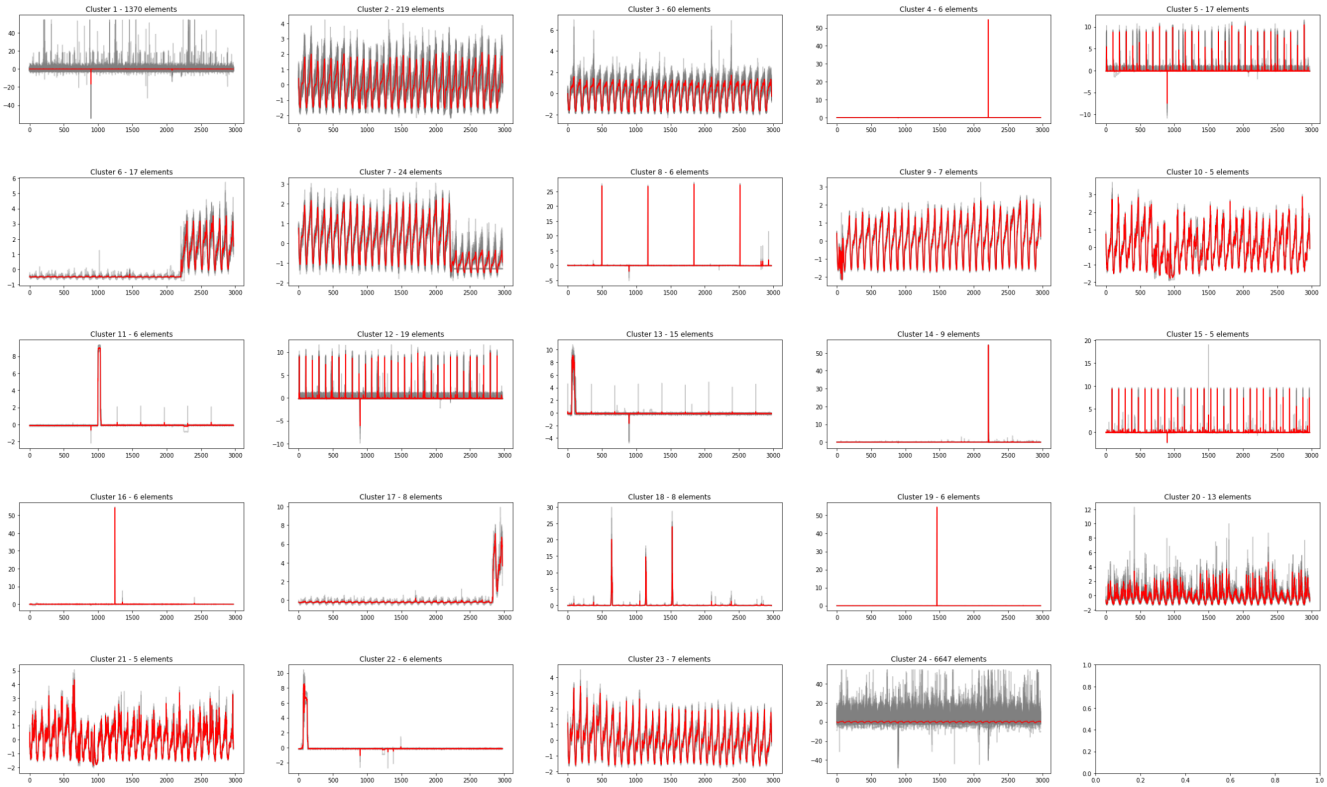


Figure A.7: Clustering using K-Means of standardized time series, transformed using PCA with 120 principal components, in 24 clusters.

A.3. DBSCAN Clustering with PCA

A.3.1. Standardized - 2 Principal Components - 0.8 eps

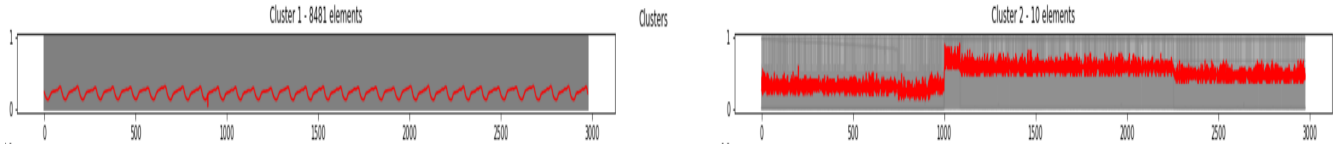


Figure A.8: Clustering using DBSCAN of normalized time series, transformed using PCA with 2 principal components, with an ϵ of 0.8.

A.3.2. Standardized - 64 Principal Components - 11 eps

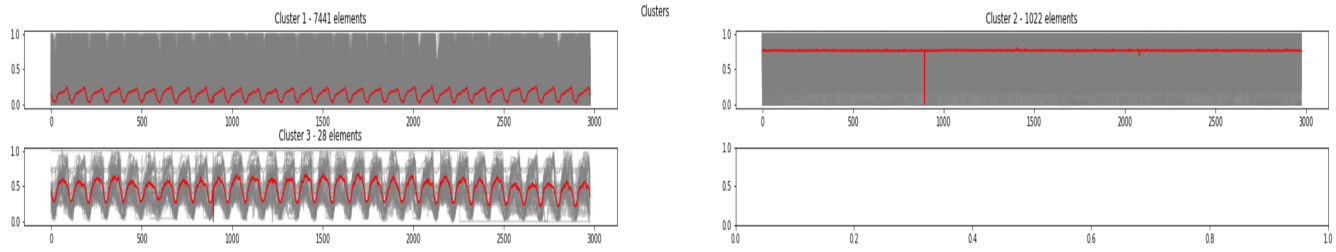


Figure A.9: Clustering using DBSCAN of normalized time series, transformed using PCA with 64 principal components, with an ϵ of 11.

B | Second Appendix - Time Series Analysis

This appendix displays the experimentation data set used in time series analysis and anomaly detection, as well as the results of the analysis itself. The plots are composed of a number of different subplots, with subplots belonging to the same network element having the same color and being boxed by black lines.

B.1. Experimentation Data Set

Each vertical set of four time series in Figure B.1 represents the six months of data for the four features of a network interface's time series. Gaps in the plots indicate missing data.

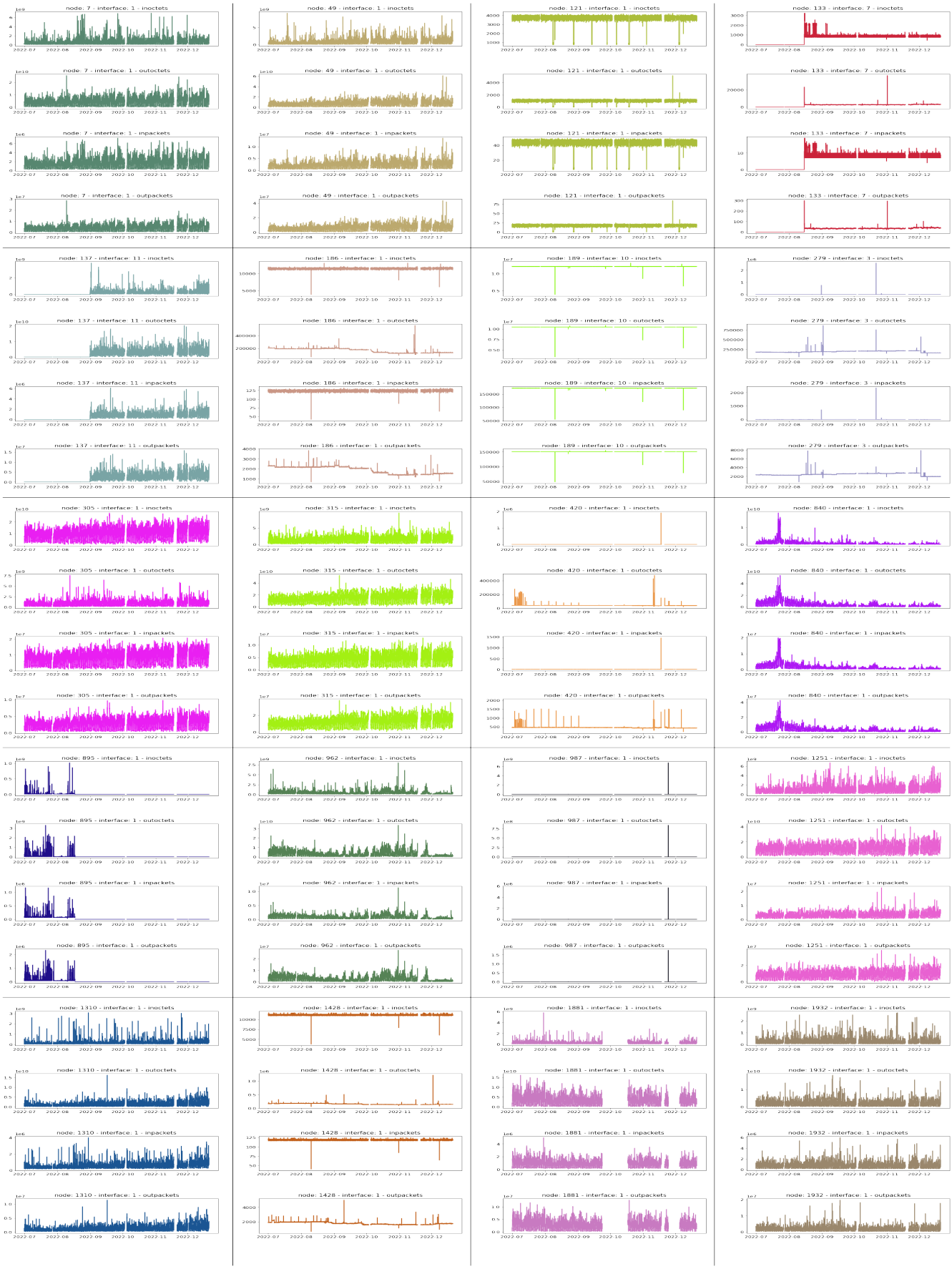


Figure B.1: Experimentation data set.

B.2. STL Decomposition

Figure B.2 displays the decomposition and reconstruction of one month of data from the first feature of each time series in the experimentation data set. The first plot displays the sequence reconstructed from the trend and seasonal components obtained through STL decomposition, superimposed on a high-transparency version of the original. The second plot displays the residual component.

Figure B.2: STL decomposition and reconstruction of the data set.

C | Third Appendix - Anomaly Detection

This appendix displays the results of the various anomaly detection approaches tested on the experimentation data set during the implementation of the thesis, as detailed in Section 4.4. The plots are etherogeneous in their style and presentation, as they were often generated using functions specific to each anomaly detection library.

C.1. STL Residuals

Figure C.1 reports the results of the anomaly detection process based on STL residuals described in Section 4.4.1. Each subplot displays the detection process being applied to one month of one time series feature, with the anomaly threshold determined through the residual component being indicated by a transparent confidence bound, and outliers being marked with red crosses. Only the first two features (`inoctets` and `outoctets`) from each network interface are reported for visualization purposes.

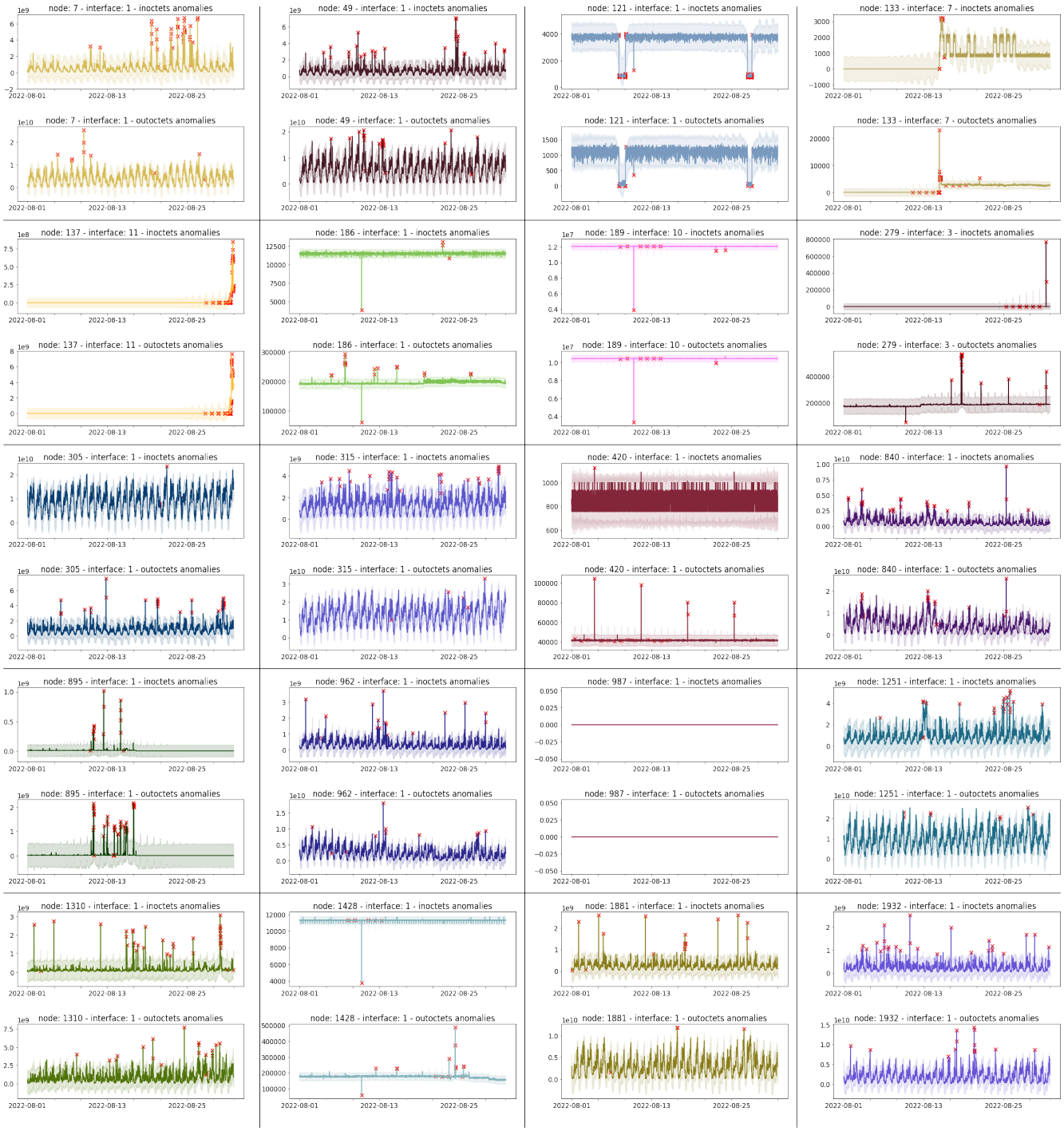


Figure C.1: Anomaly detection based on STL residuals using 3σ threshold.

C.2. Merlion

The following sections display the results of several of the anomaly detection algorithms in the `Merlion` library on the experimentation data set, as detailed in Section 4.4.4. The plots were created using `Merlion`'s built-in functionalities, and are split into two parts for ease of visualization. Each section of four plots in the figures pertains to the four features of one network interface's time series.

The plots display the original time series in black, with a vertical, dashed line indicating the separation between the model's training and test sets. In the section of the plot dedicated to the test set, a red line indicating the anomaly score of each sample is provided. A flat line represents a score of zero, indicating that the sample's anomaly score did not reach the cut-off value of the applied threshold, whereas a vertical line indicates a non-zero score of proportional magnitude. For the `Prophet` model (which is a forecasting model), the forecast is also drawn in light blue alongside its confidence interval.

C.2.1. Autoencoder

.

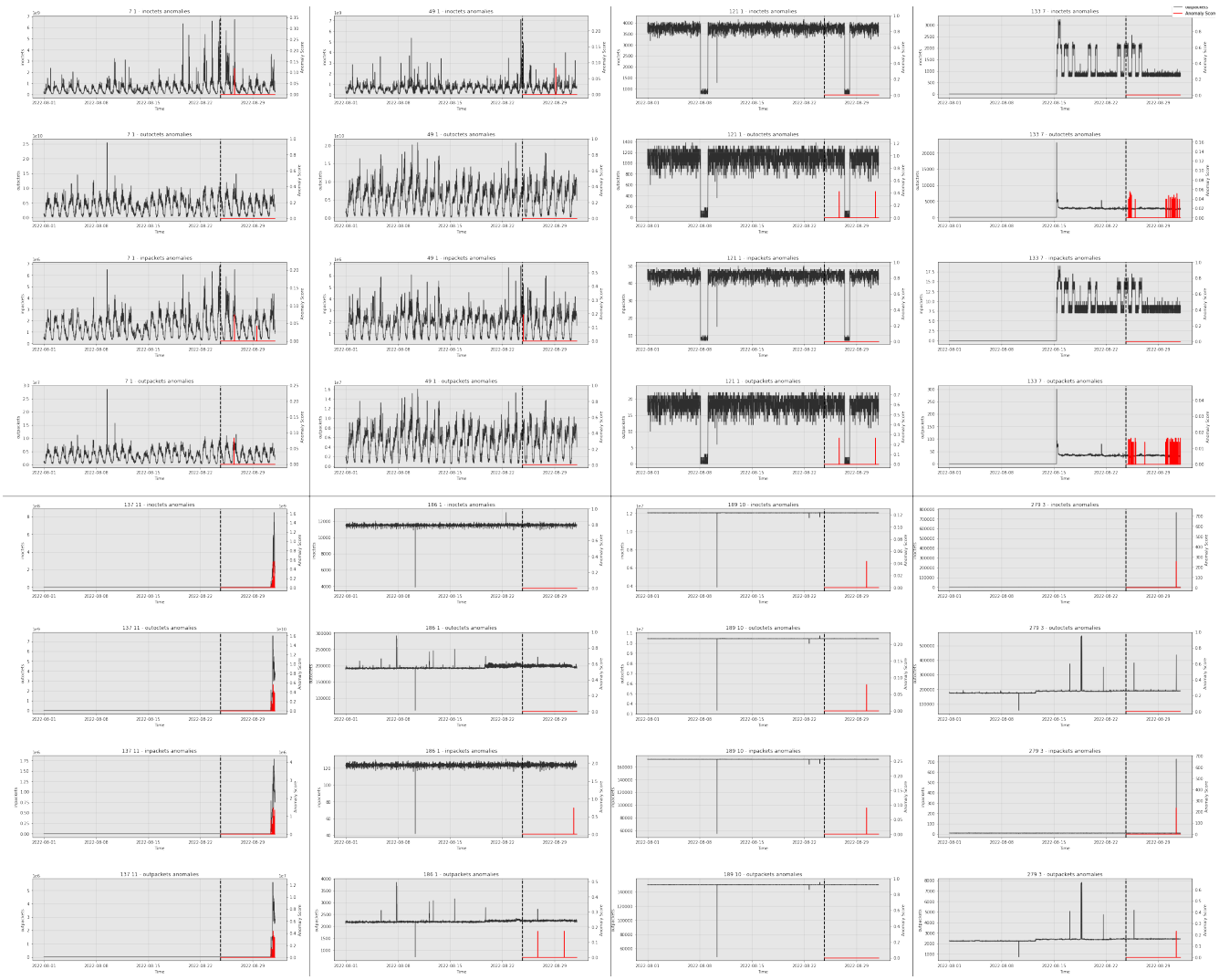


Figure C.2: Autoencoder anomaly detection - part 1.

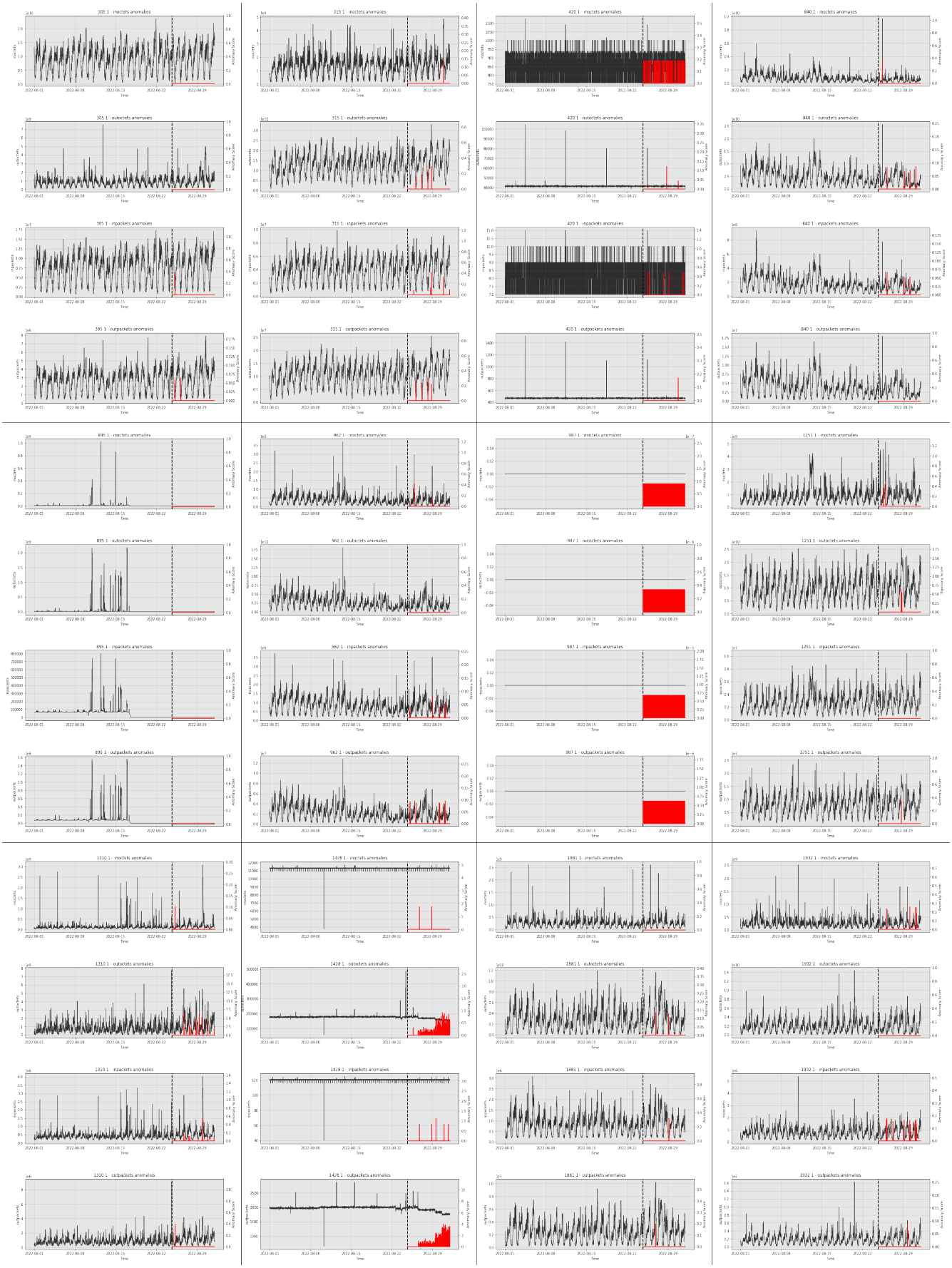


Figure C.3: Autoencoder anomaly detection - part 2.

C.2.2. DBL

.

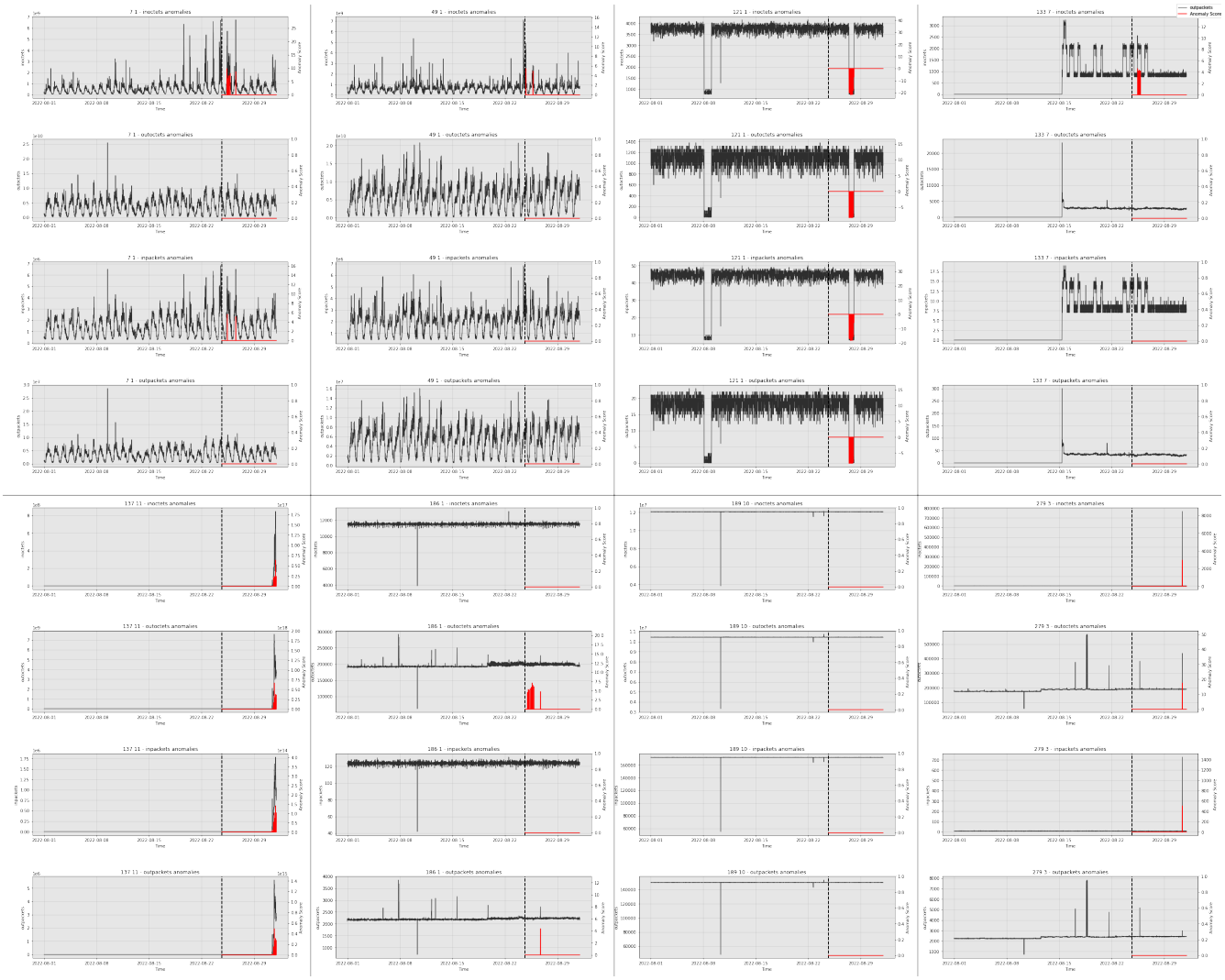


Figure C.4: DBL anomaly detection - part 1.

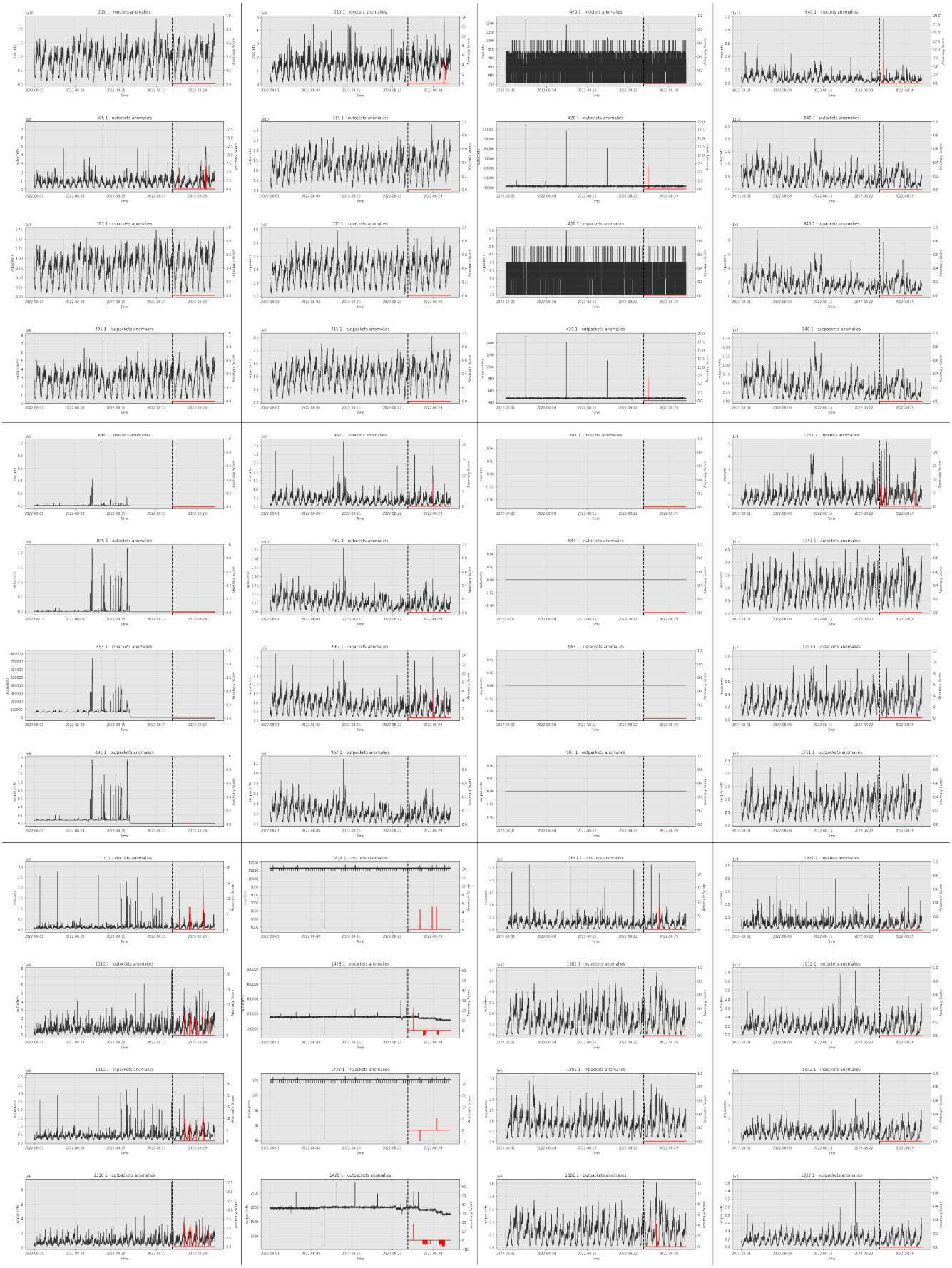


Figure C.5: DBL anomaly detection - part 2.

C.2.3. Isolation Forest

.

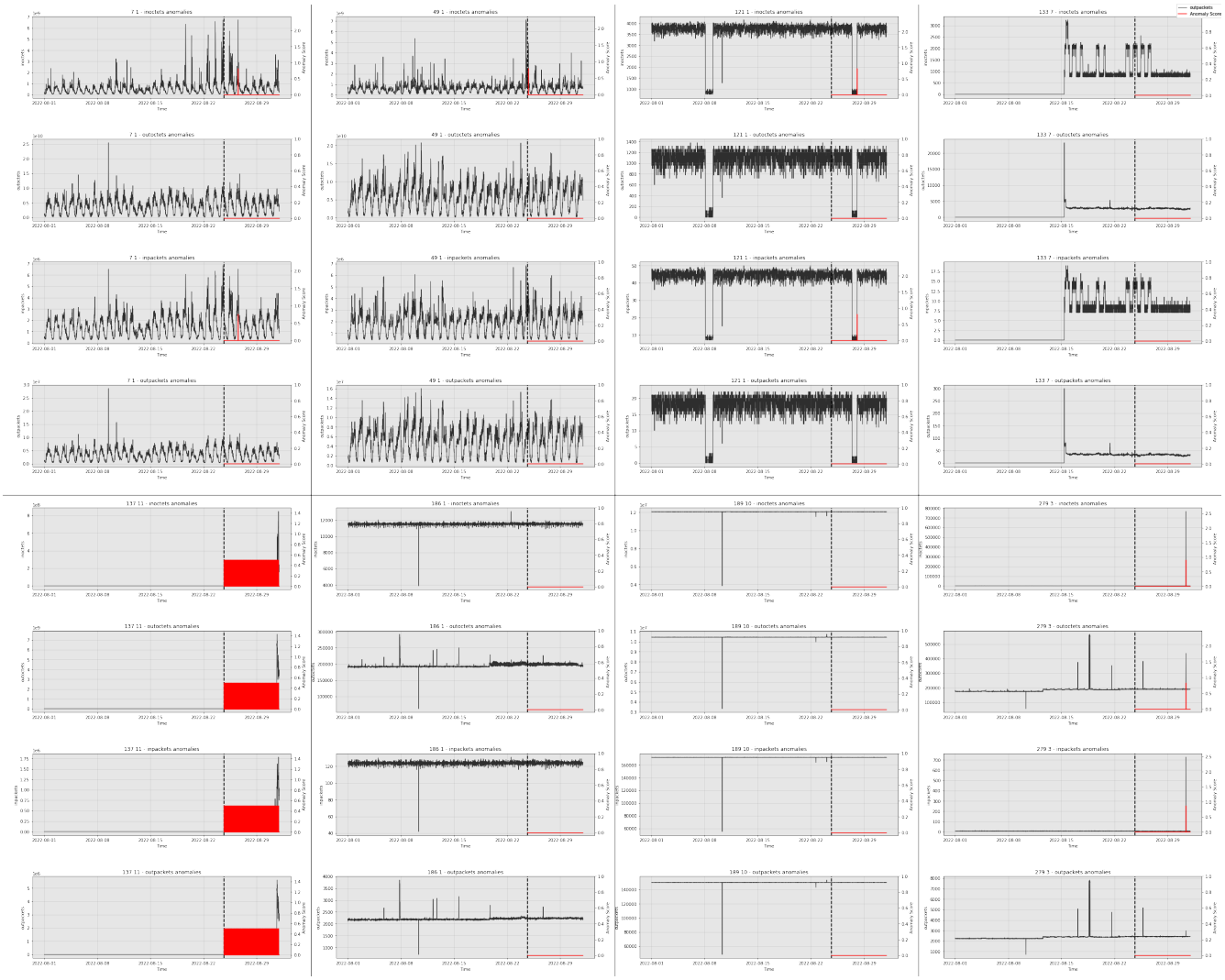


Figure C.6: Isolation Forest anomaly detection - part 1.

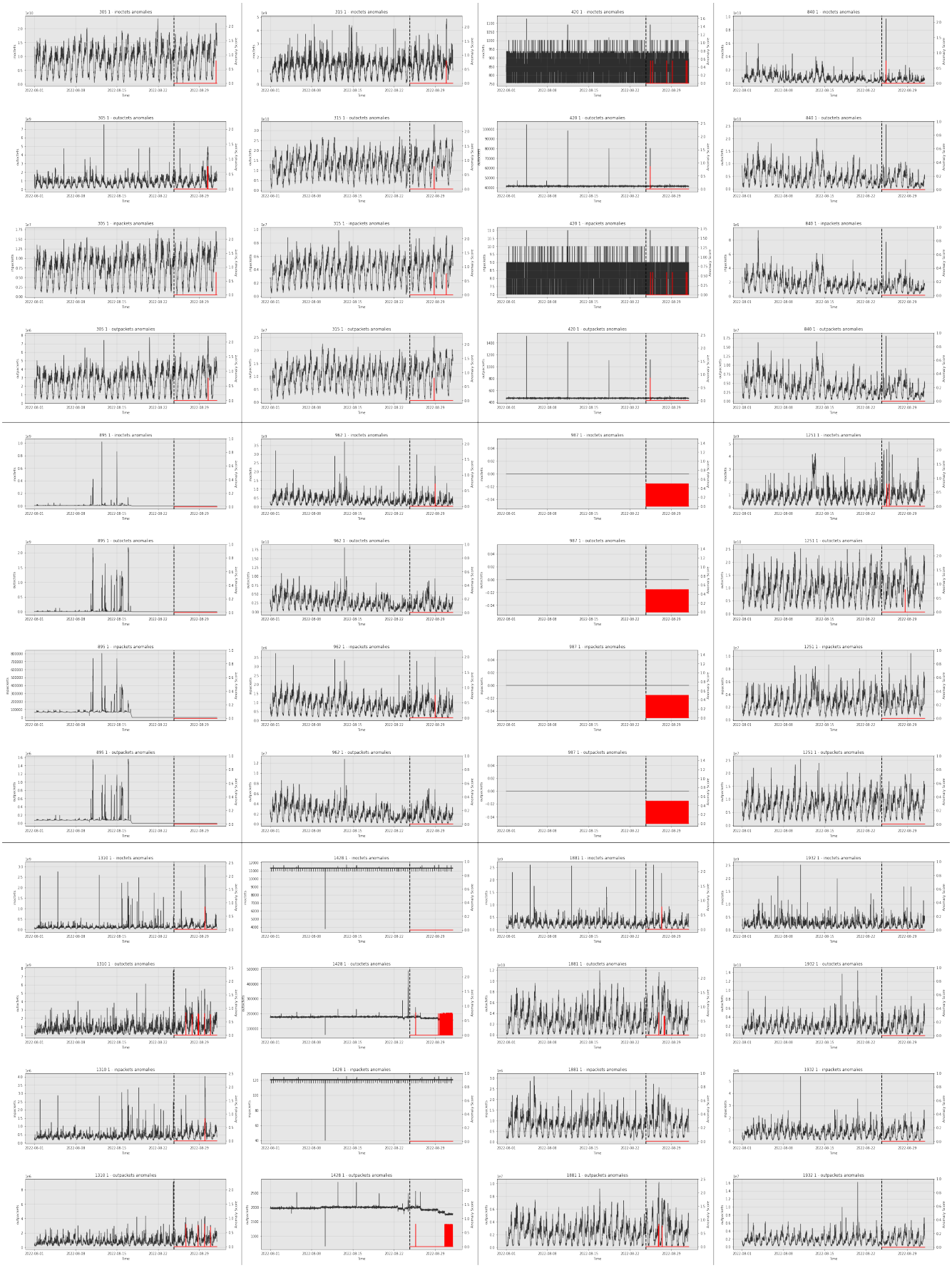


Figure C.7: Isolation Forest anomaly detection - part 2.

C.2.4. LSTM ED

.

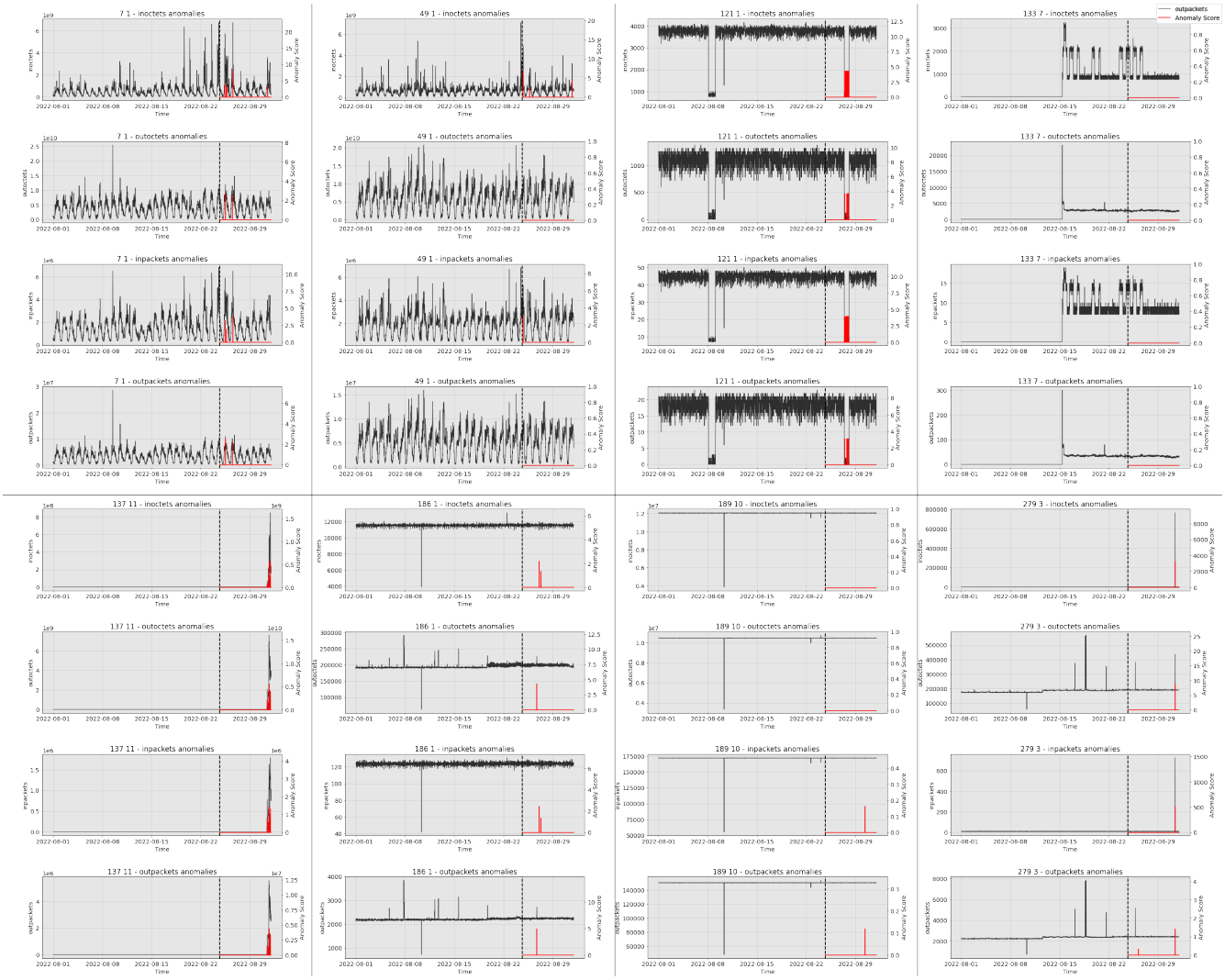


Figure C.8: LSTM ED anomaly detection - part 1.

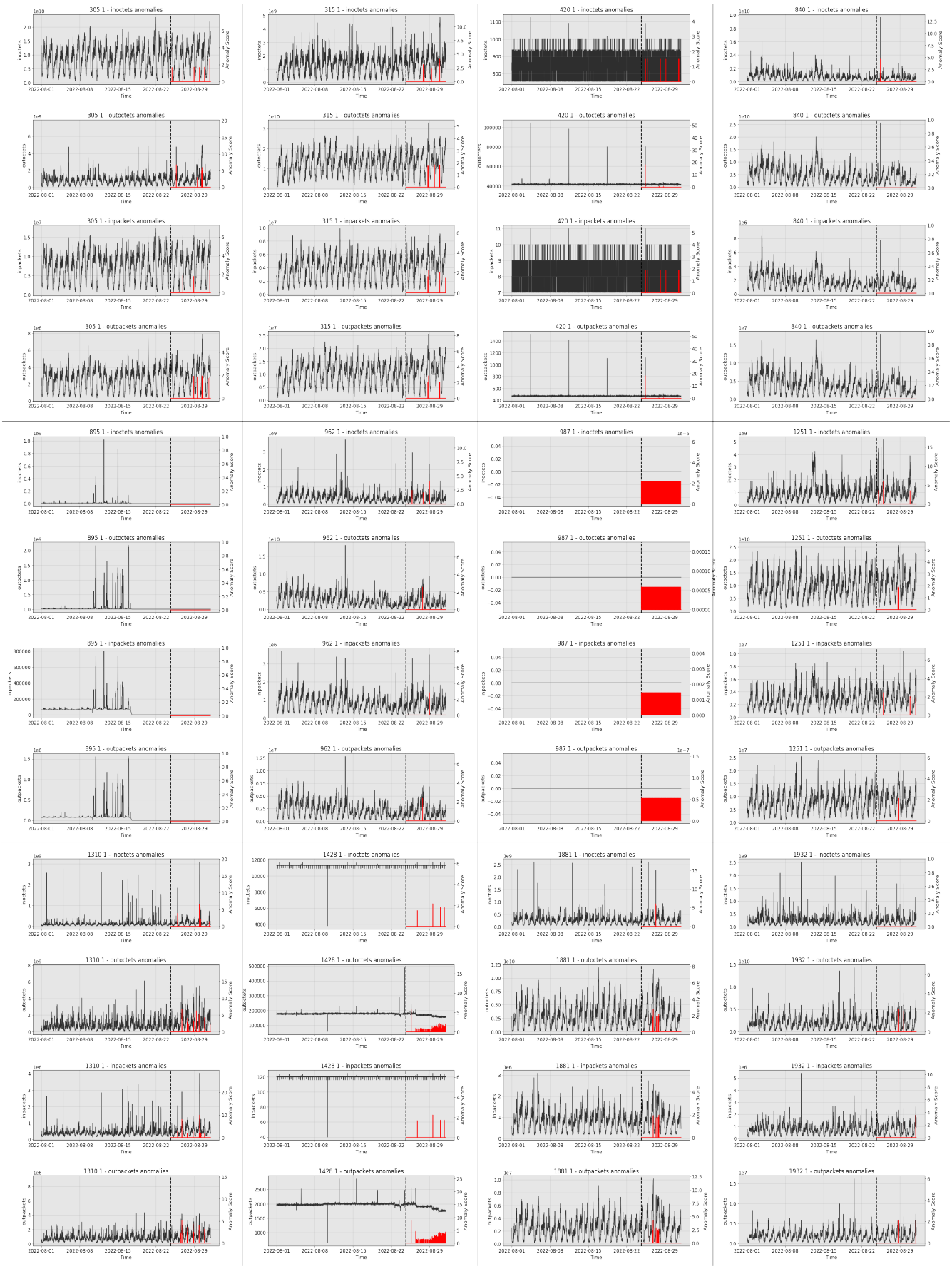


Figure C.9: LSTM ED anomaly detection - part 2.

C.2.5. Prophet

.

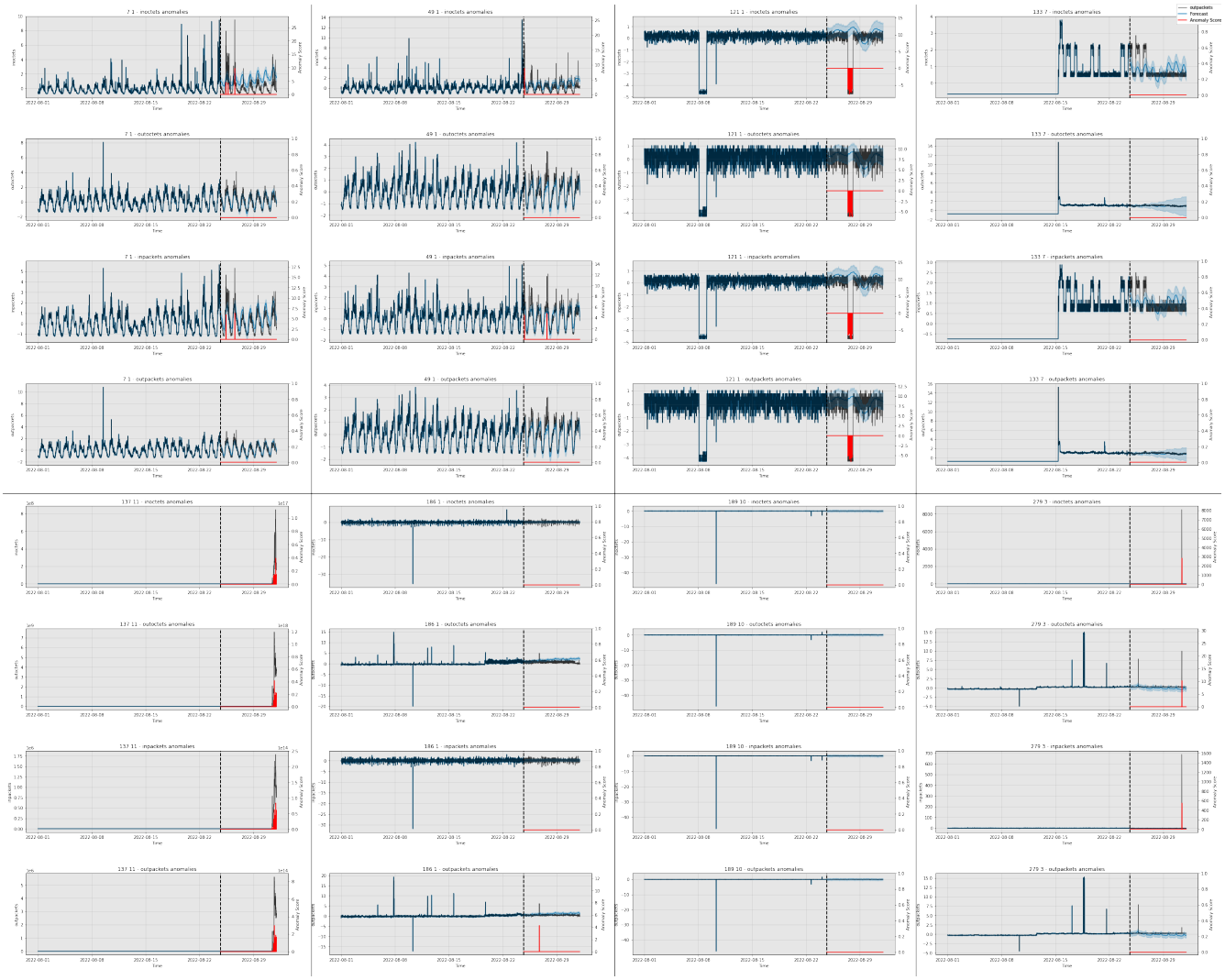


Figure C.10: Prophet anomaly detection - part 1.

Figure C.11: Prophet anomaly detection - part 2.

C.2.6. Spectral Residual

.

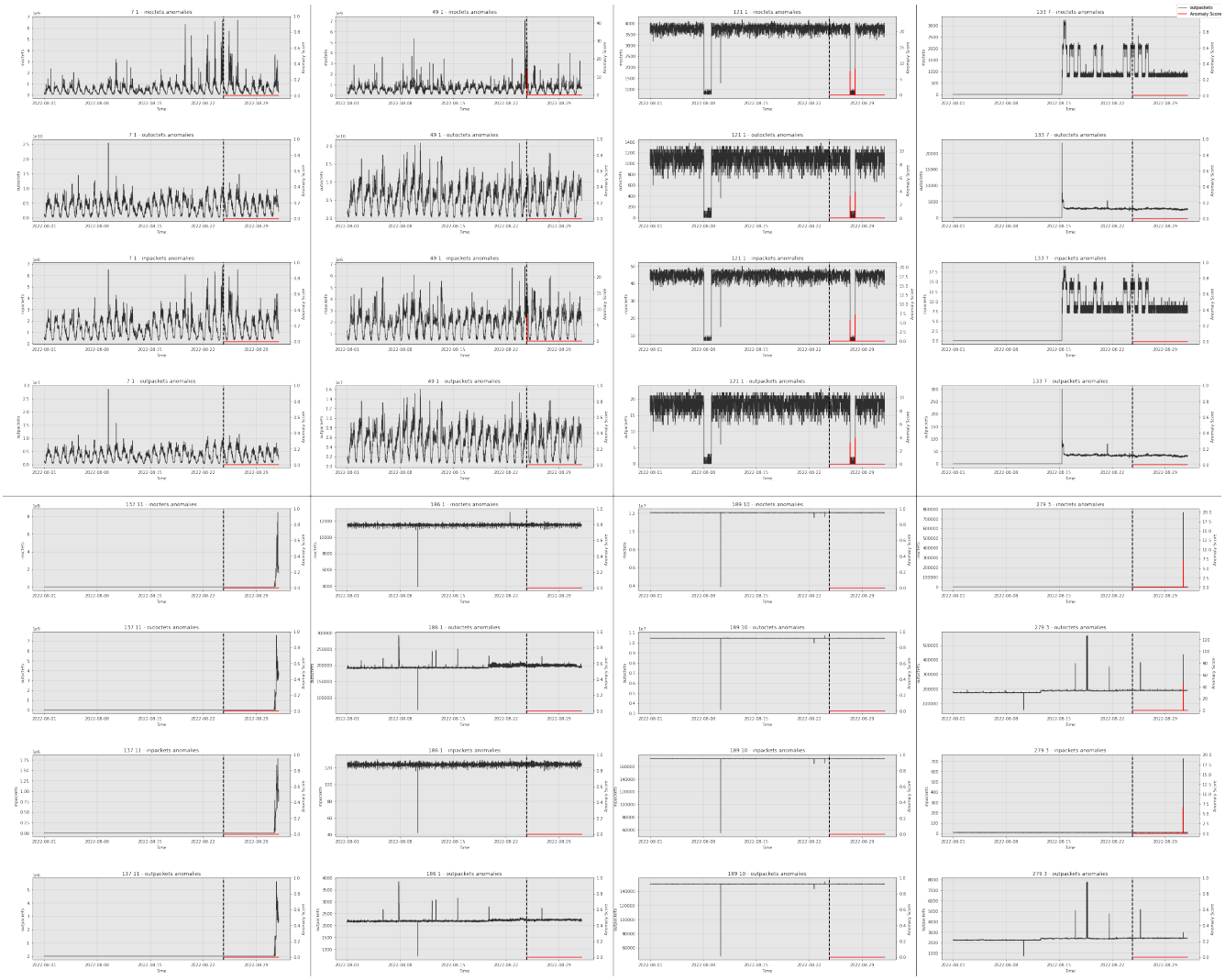


Figure C.12: Spectral Residual anomaly detection - part 1.

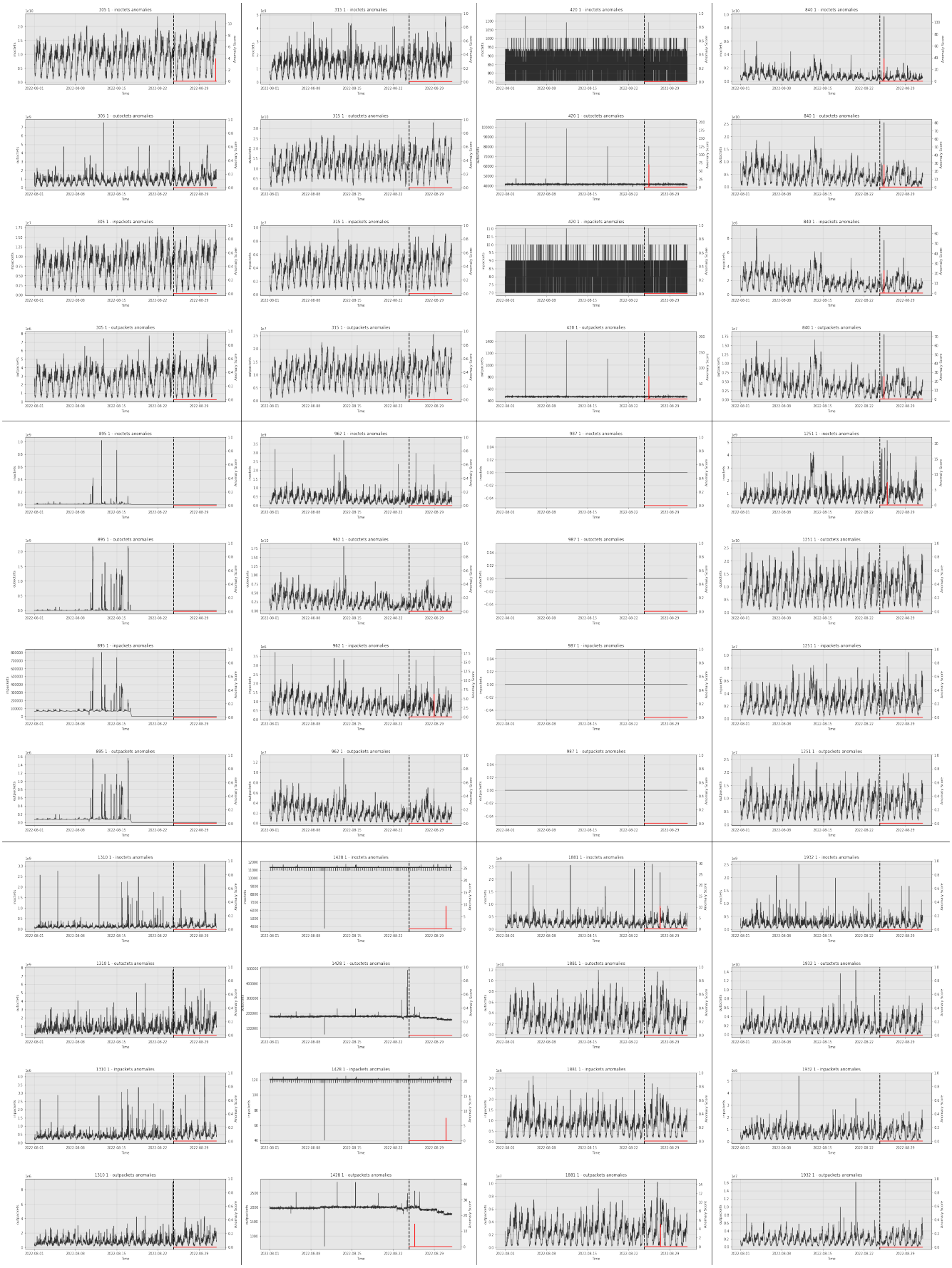


Figure C.13: Spectral Residual anomaly detection - part 2.

C.2.7. Stat Threshold

.

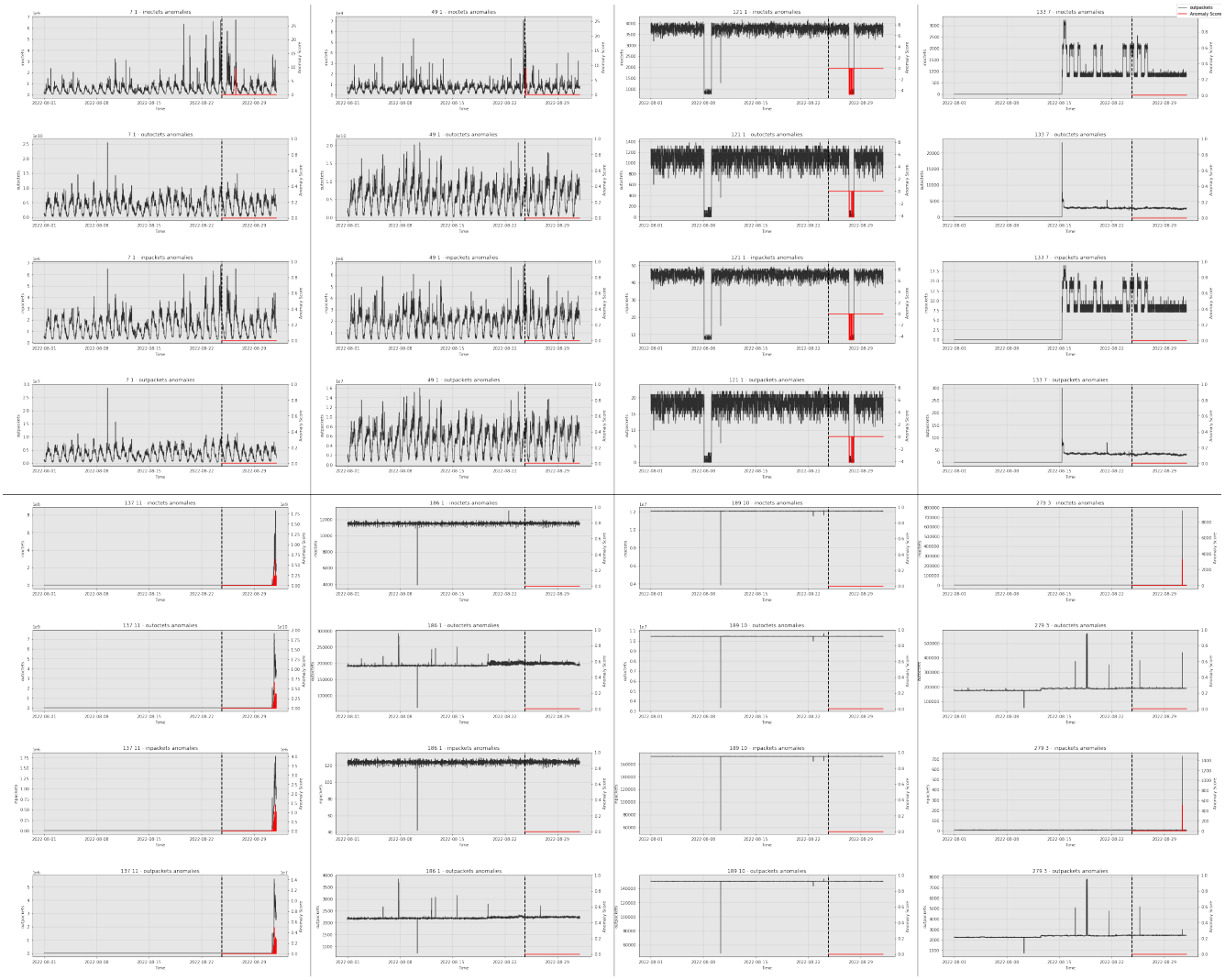


Figure C.14: Stat Threshold anomaly detection - part 1.

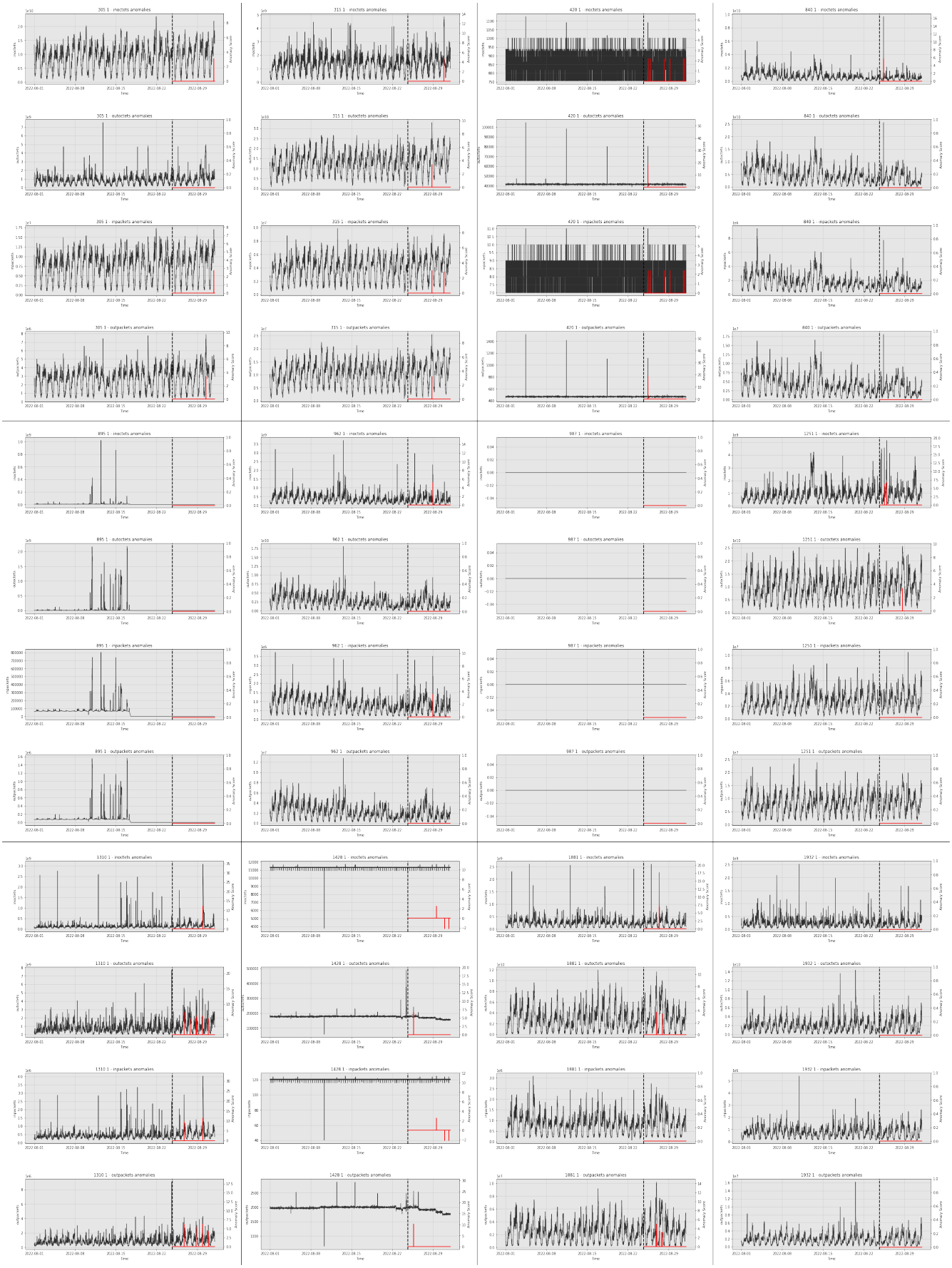


Figure C.15: Stat Threshold anomaly detection - part 2.

C.2.8. VAE

.

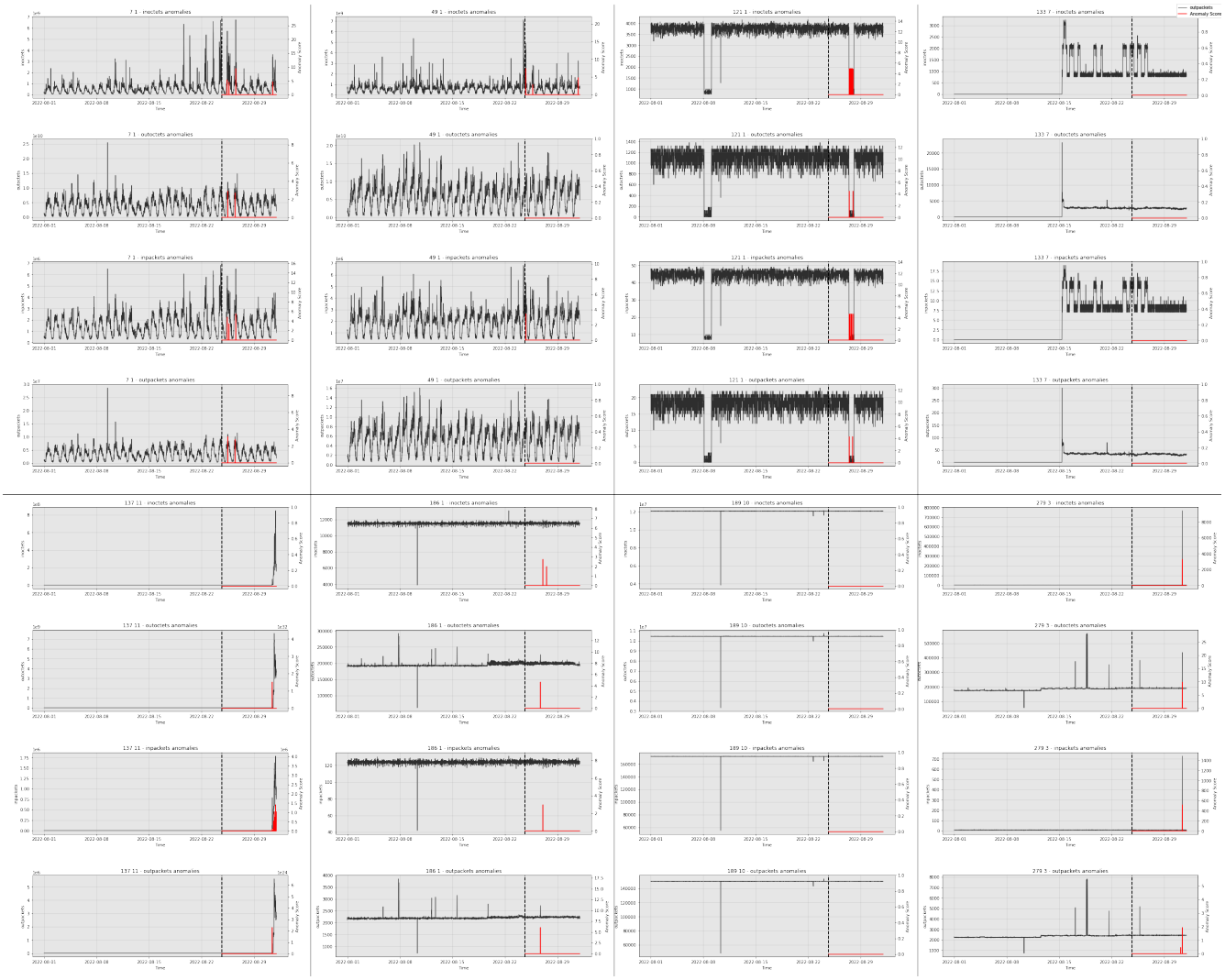


Figure C.16: VAE anomaly detection - part 1.

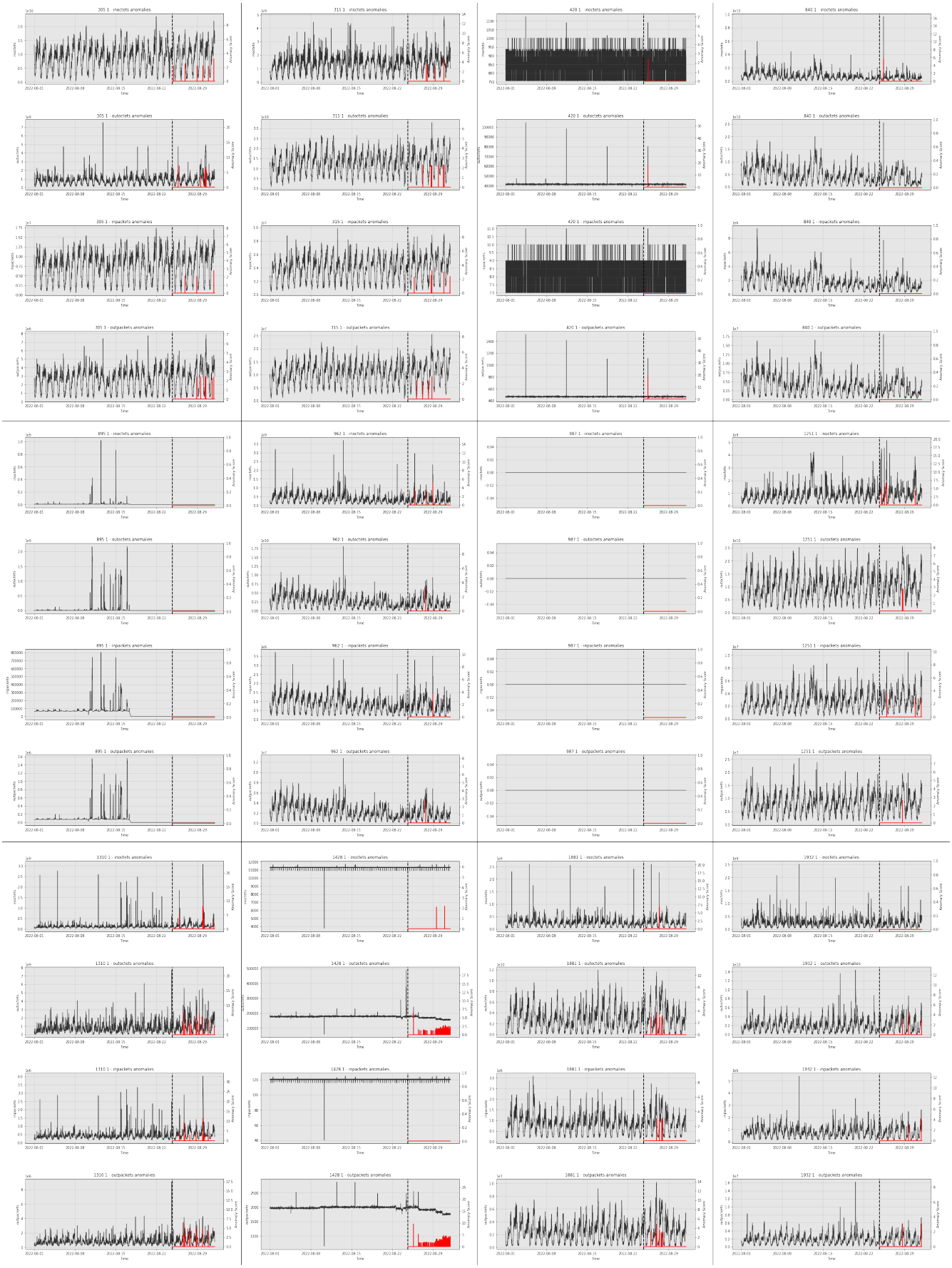


Figure C.17: VAE anomaly detection - part 2.

C.2.9. WindStats

.

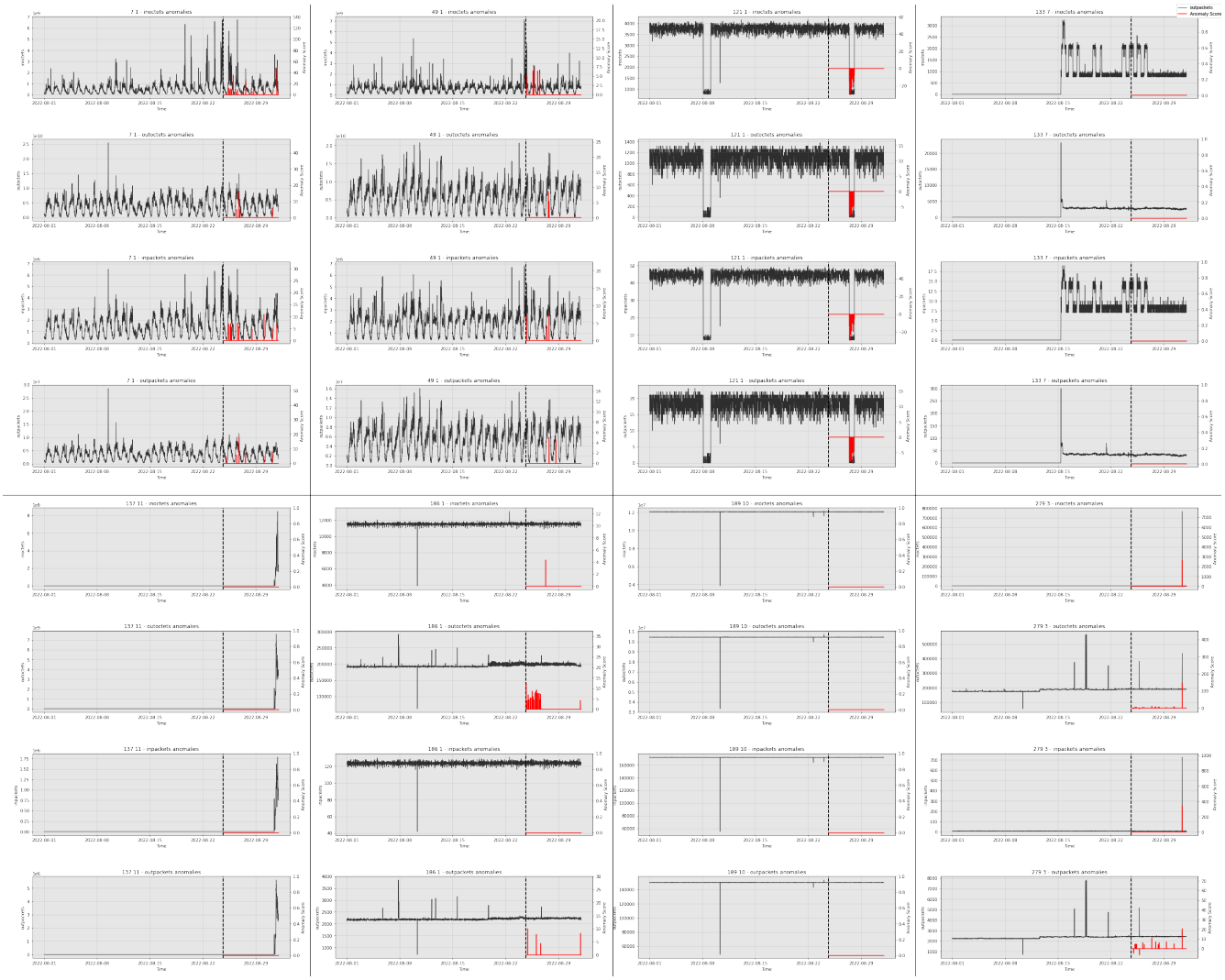


Figure C.18: WindStats anomaly detection - part 1.

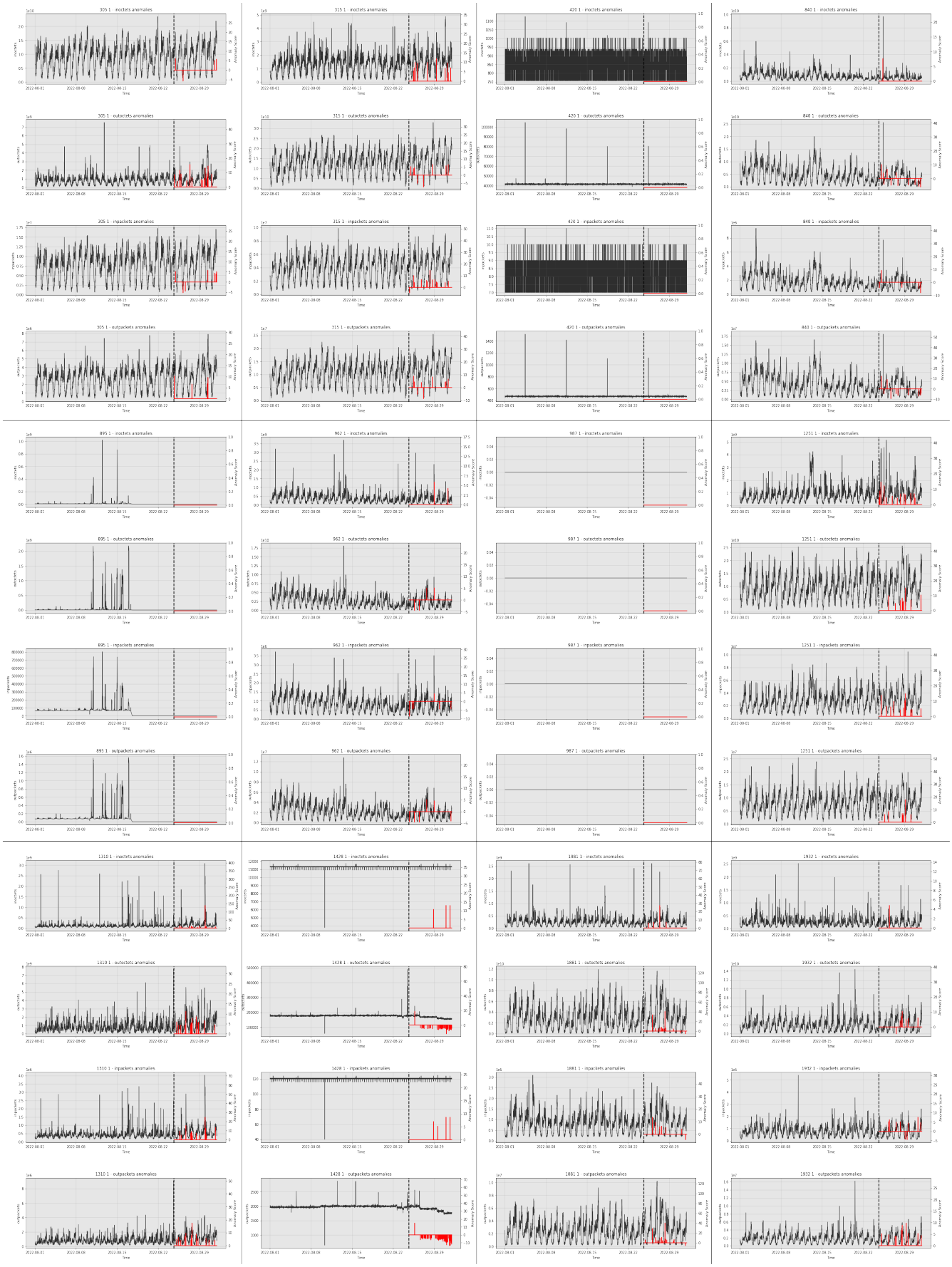


Figure C.19: WindStats anomaly detection - part 2.

C.2.10. ZMS

.

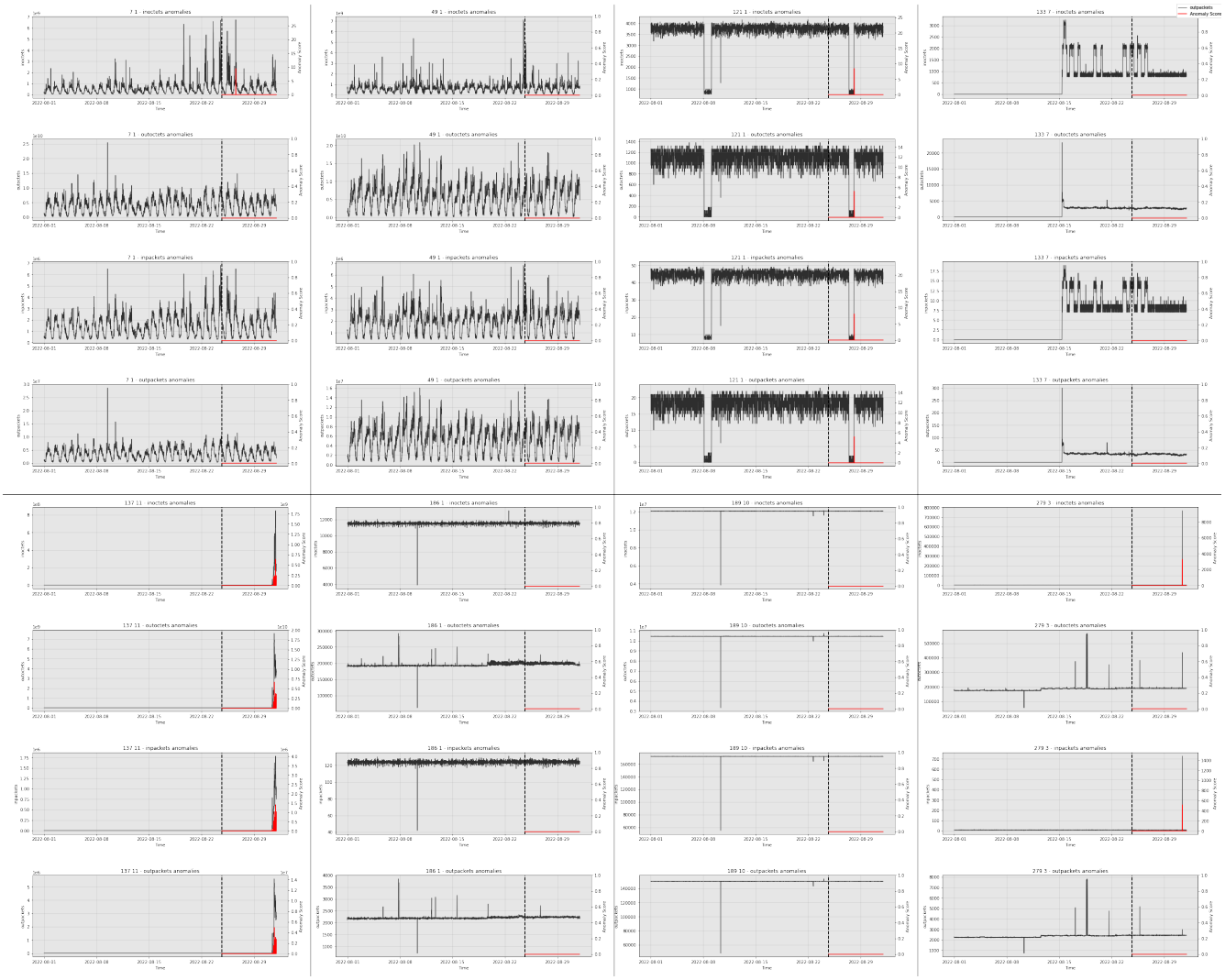


Figure C.20: ZMS

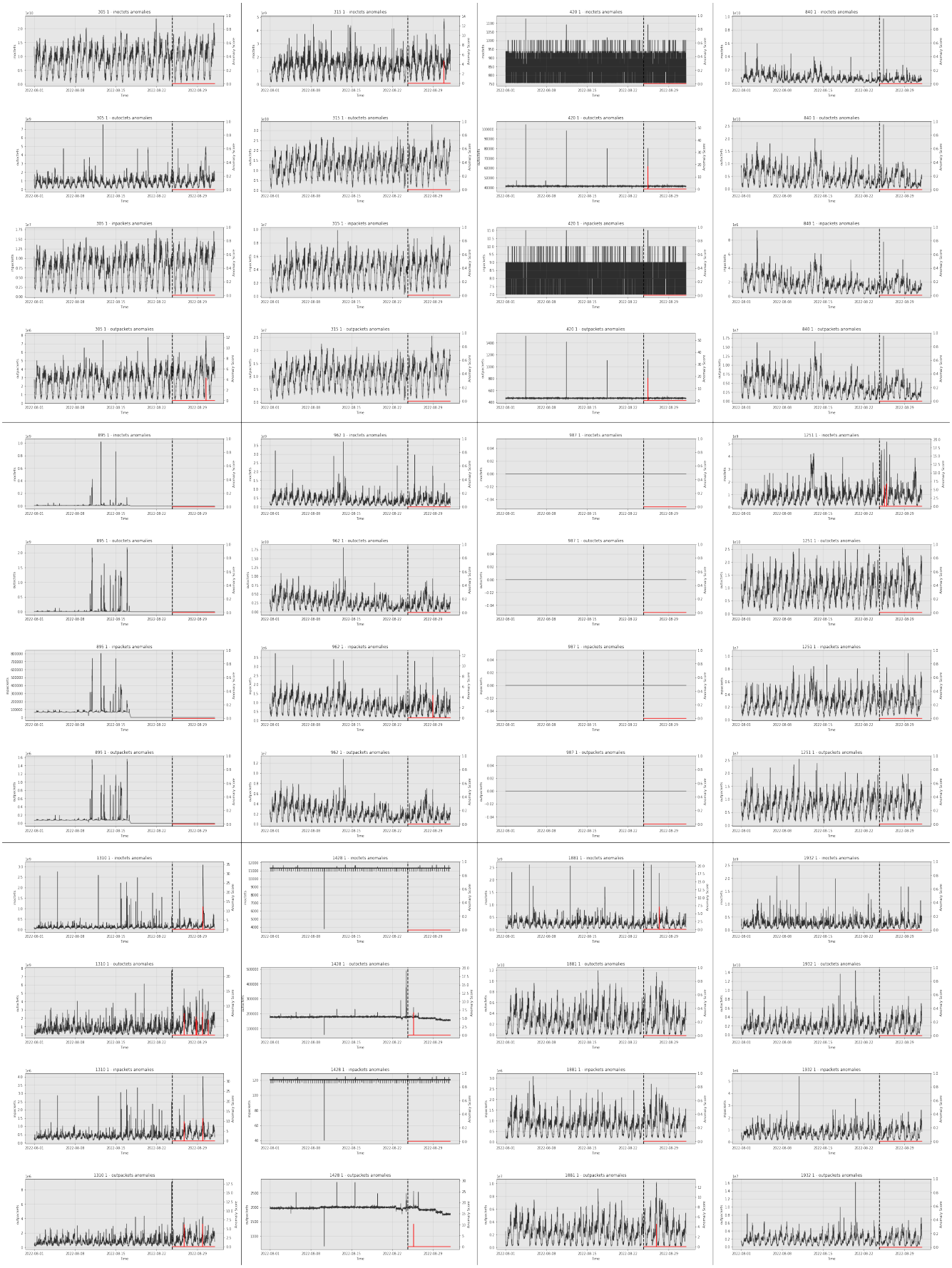


Figure C.21: ZMS anomaly detection - part 2.

Abbreviations and Acronyms

ADS	Anomaly Detection System - a system dedicated to identifying the occurrence of rare events or observations, that deviate significantly from the majority of the data and to some definition of normal behavior.
API	Application Programming Interfaces - a set of defined rules that enable different applications to communicate with each other.
DBA	DTW Barycenter Averaging - an averaging method for time series based on dynamic time warping.
DTW	Dynamic Time Warping - an algorithm often used as a time series similarity measure.
IP	Internet Protocol - the set of rules governing the format of data sent via the internet or local network.
MSA	Microservices Architecture - a type of application architecture where the application is developed as a collection of services that are independent from one another and communicate through well-defined interfaces.
PCA	Principal Component Analysis - an algorithm commonly used for dimensionality reduction, which works by finding a reduced set of variables that explain most of the variance in the data.
SDN	Software Defined Networking - an approach to design, implement, and manage networks that separates the control plane and the data plane, centralizing network intelligence in one or more network controllers.
STL	Seasonal-Trend decomposition using Loess - a technique for time series decomposition which separates time series into trend, seasonal, and remainder components.

Acknowledgements

Firstly, I would like to thank *Ericsson Finland* for providing me with the opportunity to conduct this challenging and exciting thesis project. I specifically thank Thomas Bergenwall for all of the advice he has shared with me throughout the course of this work, as well as Sauli Samila, Janne Karvonen, and Jonas Lundqvist for making it all happen.

I would also like to thank *Politecnico di Milano* for all of the precious lessons it has taught me throughout my time as a student. I thank all of the people I have met there, who have accompanied me through the highs and lows of those days, and who will doubtlessly go on to achieve great things in their lives.

I also want to thank *EIT Digital* and Federico Schiepatti, for all of the support and opportunities they provided me. I want to thank all of the wonderful friends I have met through this program, who have turned what could have simply been two normal years of university into an unforgettable and exhilarating time. And I thank Gabriele, flatmate and friend, for coming along on this adventure with me and for teaching me about *maillard*.

I also wish to thank all of the friends and loved ones who have kept me going through these stressful months, and especially my girlfriend Silvia, for the unwavering support and affection she has given me during the course of this long year, despite the distance. And lastly, I want to thank my parents and sister (and cats) for a lifetime of love and encouragement, without which none of this would have been possible.

Thank you all.

Milan, September 25, 2023

Tommaso Brumani