Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection[☆]

Max Landauer[a,*], *Markus Wurzenberger*[a], *Florian Skopik*[a], *Giuseppe Settanni*[a], *Peter Filzmoser*[b]

[a] *Austrian Institute of Technology, Austria*
[b] *Vienna University of Technology, Austria*

A R T I C L E   I N F O

Article history:
Received 5 June 2018
Revised 7 August 2018
Accepted 29 August 2018
Available online 4 September 2018

Keywords:
Log data
Cluster evolution
Anomaly detection
String clustering
Unsupervised learning
Incremental clustering
Time series analysis

A B S T R A C T

Technological advances and increased interconnectivity have led to a higher risk of previously unknown threats. Cyber Security therefore employs Intrusion Detection Systems that continuously monitor log lines in order to protect systems from such attacks. Existing approaches use string metrics to group similar lines into clusters and detect dissimilar lines as outliers. However, such methods only produce static views on the data and do not sufficiently incorporate the dynamic nature of logs. Changes of the technological infrastructure therefore frequently require cluster reformations. Moreover, such approaches are not suited for detecting anomalies related to frequencies, periodic alterations and interdependencies of log lines. We therefore propose a dynamic log file anomaly detection methodology that incrementally groups log lines within time windows. Thereby, a novel clustering mechanism establishes links between otherwise isolated collections of clusters. Cluster evolution techniques analyze clusters from neighboring time windows and determine transitions such as splits or merges. A self-learning algorithm then detects anomalies in the temporal behavior of these evolving clusters by analyzing metrics derived from their developments. We apply a prototype in an illustrative scenario consisting of a log file containing known anomalies. We thereby investigate the influences of certain parameters on the detection ability and the runtime. The evaluation of this scenario shows that 61.8% of the dynamic changes of log line clusters are correctly identified, while the false alarm rate is only 0.7%. The ability of efficiently detecting these anomalies while self-adjusting to changes of the system environment suggests the applicability of the introduced approach.

1. Introduction

Nowadays, digital systems that exist in all kinds of forms and scales are omnipresent. Despite many benefits that can be drawn from such an interconnected world, the dangers encompassed by recent technological advancements must be recognized. Larger and more complex networks generally entail the emergence of threats and novel attack vectors. Not just the amount of potential entry points becomes larger in a

☆ Massive extension of Landauer et al. (2018)
* Corresponding author.
E-mail addresses: max.landauer@ait.ac.at (M. Landauer), markus.wurzenberger@ait.ac.at (M. Wurzenberger), florian.skopik@ait.ac.at (F. Skopik), giuseppe.settanni@ait.ac.at (G. Settanni), peter.filzmoser@tuwien.ac.at (P. Filzmoser).

https://doi.org/10.1016/j.cose.2018.08.009

growing network, there is also a substantial increase of the attack surface when more complex technologies are present. This allows attackers to infiltrate the system in more diverse and previously unimaginable ways. In order to counteract such intrusions, Cyber Security employs Intrusion Detection Systems (IDS) that are able to differentiate between benign and malicious system processes and raise alerts whenever a prohibited action is executed. However, traditional IDS do so by comparing the current state of the system with known signatures. While the involved methods are usually very efficient, they fail to detect previously unknown attacks due to the fact that no corresponding entry exists in their ruleset. There is therefore a need for more flexible methods that do not rely on predefined rules derived from expert knowledge, but rather detect suspicious events occurring in large-scale ICT systems on their own.

IDS that perform such an unsupervised analysis are known as Anomaly Detection Systems and are frequently used for monitoring system logs. The advantage of log files is that they keep track of every single event that is carried out, including artifacts of attacks. An Anomaly Detection System that is able to process the log lines at least at the same rate as they appear is therefore able to detect attacks in real-time.

Due to the fact that logs are designed to be human-readable, they often contain text messages and also give information about parameters and other values related to the currently running processes. There are uncountable different ways how log files are structured in practice and the contents of most real-world log files exhibit highly different features as they depend on the type of application, configurations defining what type of messages are logged (e.g., informative messages, errors or debug output), the verbosity of the log lines, what kind of components are placed in the system and in which way they are writing their messages to the log file. Moreover, logs from many different sources are often assembled into single files or streams. For example, the syslog protocol only imposes minimal restrictions regarding the log contents when aggregating messages from different services.

This kind of content diversity apparent in many existing applications renders an automated analysis difficult and thus requires methods that provide a more flexible way of extracting relevant data out of the logs.

Several existing approaches do so by employing unsupervised or semi-supervised text clustering approaches that operate independent from the structure of the log file at hand. These methods group similar log lines into a collection of clusters, i.e., a cluster map. However, the cluster maps resulting from these algorithms usually only give a static view of the data. In general, locating outliers in these maps or single lines that contain significant words like “error” is not adequate for a thorough analysis of the system and neither is the presence or absence of certain lines sufficient to indicate problems, but rather the dynamic relationships and correlations between lines have to be considered (Xu et al., 2009).

Note that this kind of clustering is different to clustering log traces, i.e., ordered sequences of log lines, that is frequently pursued in existing literature on process mining. While the extraction of log traces requires some kind of process ID that refers to the task that generated the log messages, clustering individual log lines does not rely on any assumptions about the data. In this article, we therefore refer to static cluster maps as a collection of individual log lines rather than log sequences.

Another challenge with such static cluster maps is that they cannot be used as permanent templates for a computer system. This is due to the fact that any system generating log lines is constantly subject to changes and therefore cluster maps generated during separate time windows often turn out to consist of highly different structures. It is therefore necessary to incorporate dynamic features that span over multiple cluster maps.

This task is known as cluster evolution analysis. Fig. 1 shows an example of three cluster maps generated during three different time windows. In the first time window, the cluster map consists only of a single cluster. This cluster contains a set of log lines displayed as points and is defined by a representative, i.e., a specific element marked by a star that represents the contents of the cluster. In the second time window, two clusters exist, but only one of them is a descendant of the cluster from the first time window. This relationship between the clusters is marked by the arrow pointing from the original to the resulting cluster. In the third time window, three clusters exist, but two of them originate from a single cluster, thereby forming a split.

Cluster evolution aims at an analytical and automatic identification of such transitions between clusters. However, existing cluster evolution techniques rely on the principle that the same elements are observed and clustered over time. Log lines on the other hand are non-recurring objects, i.e., a log line occurs exactly at one single point in time and that same line is never observed again. This means that it is not possible to simply match log lines with each other without previous work, such as identifying and omitting time stamps, IDs and variable artifacts in the strings. We already mentioned that despite the fact that clustering will by definition group similar log lines into clusters, the structure and message content of lines within clusters do not necessarily have to be homogeneous. Even more so, log lines within clusters from different time windows may have structurally changed due to system events or modifications, for example, software updates that change the syntaxes of the logged messages. While fuzzy string matching algorithms exist that alleviate these issues, their extensive computational complexity in combination with the immense amount of log lines distributed in numerous clusters makes it non-trivial to determine the transitions between clusters.

Anomaly detection always relies on some kind of metric that determines whether a specific instance such as a log line, group of log lines or point in time is anomalous or not. Predefined limits are frequently used to trigger alarms for these metrics, however are not always an appropriate solution in an unsupervised setting. This is due to the fact that different systems usually show highly different behavior and also the behavior of a single system changes over time. A self-learning procedure should therefore be able to dynamically adjust to any environment it is placed into and adapt the limits for triggering alarms on its own.

Finally, an anomaly detection system that deals with all the previously mentioned issues must also exhibit a reasonable computational complexity regarding runtime and

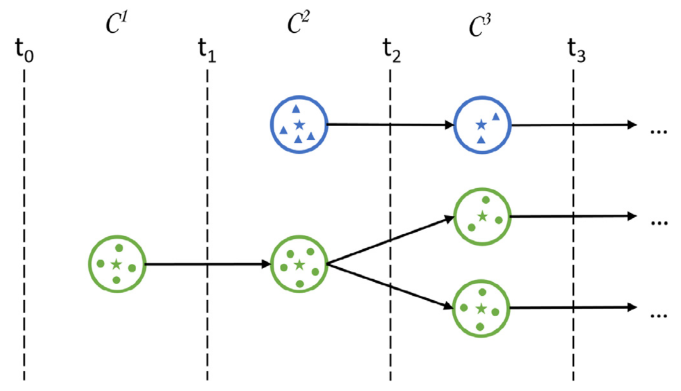


Fig. 1 – Example of cluster evolutions spanning over 3 time windows.

memory consumption. Due to the fact that online anomaly detection is supposed to take place in real-time, the algorithm needs to be efficient enough to process log lines at least as fast as they appear. Furthermore, it must be ensured that the used methods are suited for the processing of streams, i.e., the runtime should scale linearly with the number of log lines and there must be a limit to the required memory.

The approach proposed in this article solves the aforementioned issues by encompassing the following contributions:

1. A clustering model that is able to connect log line clusters from a sequence of static cluster maps and thereby supports the detection of transitions between these clusters,
2. the definition of metrics that are derived from aforementioned transitions between clusters,
3. an online anomaly detection approach that displays the security-relevant metrics as time-series and employs forecasting models in order to detect deviations from expected behavior,
4. an evaluation of the introduced methodology by deploying the prototype in a realistic scenario.

The main feature of the introduced approach is that contextual anomalies, i.e., log line types that do not cohere to previously gained knowledge about their average frequency of occurrence, periodicity and correlation, are detected. This extends the ability of static clustering approaches that detect highly dissimilar lines which occur only once as outliers rather than temporal anomalies which are observed as system behavior changes over time. Moreover, the introduced approach is self-learning and does not require any previous knowledge about attacks or the structure and content of the log data. This allows the handling of complex log lines from any number of processes and components in arbitrarily formats and appearances following different standards or no standards at all.

This article is a massive extension of our approach proposed in Landauer et al. (2018). In general, for this article we largely increase the level of detail of our work by elaborating on the proposed concepts more precisely. We thereby outline new ideas of approaching cluster evolution, e.g., by proposing a sophisticated multi-window model. Furthermore, we include additional evolution metrics and go into algorithmic details. We largely extend the evaluation to comprise discussions about parametric influences and types of detected anomalies. Finally, in this article we also apply our algorithm on real log data in addition to semi-synthetically generated data.

The remainder of this paper is organized as follows: Section 2 surveys existing approaches for anomaly detection and cluster evolution. For a better understanding of subsequently introduced concepts, Section 3 illustrates the procedure of the proposed method with the aid of an example. Section 4 then goes into detail about the incremental clustering algorithm. Section 5 extends on the clustering algorithm by introducing a clustering model that supports cluster evolution techniques. Metrics derived from the evolutions are then used for anomaly detection using time-series prediction in Section 6. The theoretically discussed models are then applied within a realistic scenario in Section 7. Finally, Section 8 concludes the paper and further states suggestions and ideas for future research in this topic.

2. Related work

The high risk posed by cyber threats has led to a massive interest in securing computer systems. Accordingly, a vast amount of research in the field of cyber security has been carried out and there exist numerous works focusing on anomaly detection. Many approaches employ unsupervised algorithms that due to the fact that they do not require previous knowledge about the data and implicitly assume that outliers only make up a small part of the input data. These are clear advantages over supervised methods that require labeled data sets, i.e., a log file with annotated lines that is used as a training set. Creating such labeled log files usually involves the time- and resource-consuming manual work of experts and only provides little benefit regarding the detection of unknown attacks.

Independent of the learning method, Chandola et al. (2009) differentiate between three types of anomalies: (i) point anomalies, i.e., outliers that are dissimilar to all other data points, (ii) contextual anomalies, where the context could be defined by a current system state or time, and (iii) collective anomalies, i.e., anomalous groups of data points. For detecting point anomalies in log files it is usually not necessary to take dynamic features of the data, such as temporal correlations, chronological orderings and occurrence frequencies, into account. The log data may be represented in high-dimensional spaces, where outliers are detected using dimension reduction techniques Juvonen et al. (2015). Other approaches frequently apply probability theory Kruegel and Vigna (2003) and Bayes Statistics Amor et al. (2004) in combination with clustering techniques Yassin et al. (2013) in order to analyze network traffic for outliers. Such clustering methods are also employed for the creation of event signatures or templates that are then used to group log lines and thereby identify outliers. For example, the algorithm SLCT introduced by Vaarandi (2003) generates patterns by observing frequent words and their respective positions in each line.

In order to employ such log line templates for dynamic anomaly detection, He et al. (2016) make use of an event count matrix and determine whether there are deviations from the normal amount of occurrences of certain log line types. Such message counts were also used by Xu et al. (2009) who used static source code analysis for the generation of log templates. Fu et al. (2009) also used log patterns in combination with a Finite State Automaton and considered transit times and circulation numbers as measures to detect anomalous system behavior. In order to circumvent the necessity of log templates, Andreasson and Geijer (2015) considered string metrics and n-gram matching for clustering. Finally, incremental cluster methods are able to dynamically add any number of incoming data points by either allocating them to one of the existing clusters or declaring them as outliers if the distance to the nearest cluster exceeds a certain threshold. Such an incremental approach has been applied for log file anomaly detection on systems with a highly predictable behavior and a large number of repeating sequences (Wurzenberger et al., 2017).

A different but increasingly popular method for learning patterns in data is posed by neural networks. Multilayer perceptrons and time-series analysis methods are used by Hill and Minsker (2010) and Cortez et al. (2012). Long Short Term Memory Recurrent Neural Networks are further able to detect temporal dependencies in event logs (Goh et al., 2017). Also Restricted Boltzmann Machines have been used for anomaly detection in network data (Fiore et al., 2013).

A large amount of research focuses on time series analysis for clustering (Esling and Agon, 2012; Khalilian and Mustapha, 2010; Silva et al., 2013) and anomaly detection (Chin et al., 2005; Gupta et al., 2014; Pincombe, 2005; Sperotto et al., 2008; Thottan and Ji, 2003) in network traffic data. Cluster evolution techniques thereby pose a feasible alternative for monitoring the developments of log clusters over time.Spiliopoulou et al. (2006) propose an algorithm for the identification of transitions between clusters necessary for such a reasoning. Smoothing the identified transitions has shown to be advantageous as anomalies may negatively influence the quality of detected cluster evolutions Chi et al. (2009). Moreover, the structures of previously generated cluster maps should be taken into account when clustering in order to ensure that their differences are minimized (Chakrabarti et al., 2006).

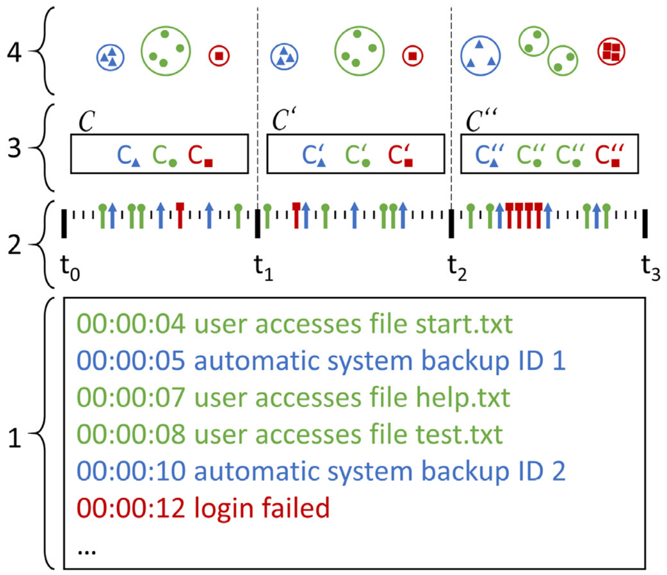
Not only the internal compositions of clusters, but also their relationships to each other as well as their correlations are of relevance for anomaly detection. This also includes the relative position and movements of the clusters (Carmi et al., 2009), which is a problem also found in GPS tracking (Jensen et al., 2007). Toyoda and Kitsuregawa (2003) specify several additional internal and external cluster metrics related to cluster evolution analysis. These cluster properties may also be measured using sliding window approaches (Zhou et al., 2008).

When artifacts related to network connections such as IP addresses can be derived from log lines, cluster evolution enables anomaly detection on graphs (Asur et al., 2009; Bilgin and Yener, 2006; Bródka et al., 2013; Chan et al., 2008; Falkowski et al., 2006; Lee et al., 2014). However, log lines are non-recurrent entities, i.e., a specific log line only occurs at one point in time and cannot simply be related to any other line occurring afterwards. Thus, existing graph-based cluster evolution techniques cannot be applied on raw log data. Moreover, most cluster techniques do not support dynamic changes that are caused when inserting new data points to an existing cluster map, but rather require a complete reformation of all clusters. The approach proposed in this work therefore tries to solve these issues by connecting clusters of log lines that were generated within subsequent time windows.

Overall, the anomaly detection methodology proposed in this article differs from all mentioned works regarding several aspects. First, it is an approach for dynamic and contextual anomaly detection rather than the detection of point anomalies. Second, our clustering approach does not rely on source code analysis, neural networks, or log templates for pattern matching, but employs flexible string distance metrics that operate on characters rather than words. Third, our approach makes use of cluster evolution in order to process log data in a streaming manner rather than being limited to fixed-size data sets. Finally, while many existing solutions are not adaptive to system modifications, our approach incorporates these changes in its anomaly detection procedure. Given these features, we argue that our proposed approach is a valuable contribution to the current state of the art.

3. Concept

In this section, an illustrative example is discussed in order to explain the difficulties of performing cluster evolution on raw log data, motivate the advantages of dynamic log file analysis and outline the proposed concept that will be explained in more detail in the following sections. Fig. 2 shows this example. In the bottom part marked with 1, several log lines are displayed with preceding time stamps. At this point the lines have no relation to each other, the colors and marks were only added for a better visualization of the groupings that will be determined in the clustering step. In the example, three processes ◯, △ and □ produce specific types of log lines, i.e., process ◯ logs user file accesses that appear in random intervals, process △ logs an automated backup procedure that generates lines in regular intervals and □ logs failed login attempts. In step 2, the occurrences of the lines are displayed on a time axis, where t_0, t_1, t_2, t_3 represent the boundaries of the time windows of each cluster map. Step 3 then clusters the lines according to a given clustering algorithm and string similarity metric. This results in the three cluster maps $\mathcal{C}, \mathcal{C}', \mathcal{C}''$ from three different time windows. As it can be seen, while in the



```
00:00:04 user accesses file start.txt
00:00:05 automatic system backup ID 1
00:00:07 user accesses file help.txt
00:00:08 user accesses file test.txt
00:00:10 automatic system backup ID 2
00:00:12 login failed
...
```

Fig. 2 – (1) Log file. (2) Log events occurring within time windows. (3) Static cluster maps for every time window. (4) Schematic clusters undergoing transitions.

first two time windows only 3 clusters were found, the cluster map of the final time window consists of 4 clusters. Without cluster evolution it would be hardly possible at this point to determine how the formation of those 4 clusters was accomplished or how any of the clusters from different time windows relate to each other. However, the graphical display of the result of a cluster evolution marked with 4 in the top part of the figure gives several useful insights: Not only is it possible to track the clusters over all time steps, it can be seen that cluster $C_{\bigcirc}$ splits up in the last time step. Moreover, cluster $C_{\triangle}$ increases its distance to the other clusters in $\mathcal{C}'$ and further becomes more diffuse in $\mathcal{C}''$ which is represented by the diameter of the circle. Finally, it can be seen that $C_{\square}$, which represented an outlier in $\mathcal{C}$ and $\mathcal{C}'$, i.e., the line could not be allocated to any other group of messages and thus remained alone in its own cluster, increased its size in $\mathcal{C}''$. All of these effects are indicators for abnormal behavior, for example, the increase of lines that lie inside the 'login failed' cluster may be caused by an attempt to break into a user account by a brute force attack.

It can therefore be concluded that not only single log lines that do not match any of the existing clusters are of relevance when searching for abnormal system behavior, but also that the properties of the clusters themselves viewed over time have to be considered as indicators for anomalies. It is however not trivial to derive any insights from clusters of separate time windows since they are generated by different sets of log lines. Accordingly, anomalies regarding the temporal dependencies such as log line frequencies, periodical behaviors or correlations cannot be detected. We therefore propose a solution to this problem by linking clusters throughout multiple time windows in order to generate time-series from their evolutions.

Our approach that was first published in Landauer et al. (2018) consists of several steps that involve techniques from clustering, cluster evolution and time-series analysis. A detailed overview about the steps of the algorithm is given by the flowchart in Fig. 3. Steps (1)-(4) describe the clustering procedure that is discussed in detail in Section 4. The log lines read from the log file or stream in step (1) are one after the other transferred to the preprocessing stage in step (2). There, special characters may receive specific treatment in order to facilitate clustering, but the overall structure and content of the log lines remains the same. For example, characters outside of the range $32 - 126$ from the ASCII table may be removed, multiple consecutive spaces may be reduced to a single space and time stamps may be extracted. The cluster map is then iteratively build in step (3) by adding each log line to one of the existing clusters or generating a new cluster if necessary. After the clustering procedure has been carried out within a number of time windows, an ordered sequence of static cluster maps is established. As already mentioned, each log line only occurs in one specific time window and can therefore not be related to a cluster from a different time window. We therefore introduce an allocation phase in step (4) that establishes this connection by allocating the log lines in the cluster maps of the preceding and succeeding time windows.

After the completion of each time window, step (5) makes use of the log line allocations resulting from the previous step and determines which cluster from the current cluster map originates from which other cluster from the preceding cluster map. In addition, advanced transitions such as splits or merges between clusters are detected and appropriately handled. The procedure of allocating log lines to neighboring cluster maps and the identification of transitions between clusters are explained thoroughly in Section 5. Based on the identified transitions, evolution metrics that measure the state of the cluster map and indicate changes of specific clusters are computed in step (6). For example, an evolution metric could be used to determine how many log lines were allocated to this cluster or whether a cluster is stable, i.e., whether the log lines allocated to this cluster in one time window were also allocated to the same cluster in another time window. In step

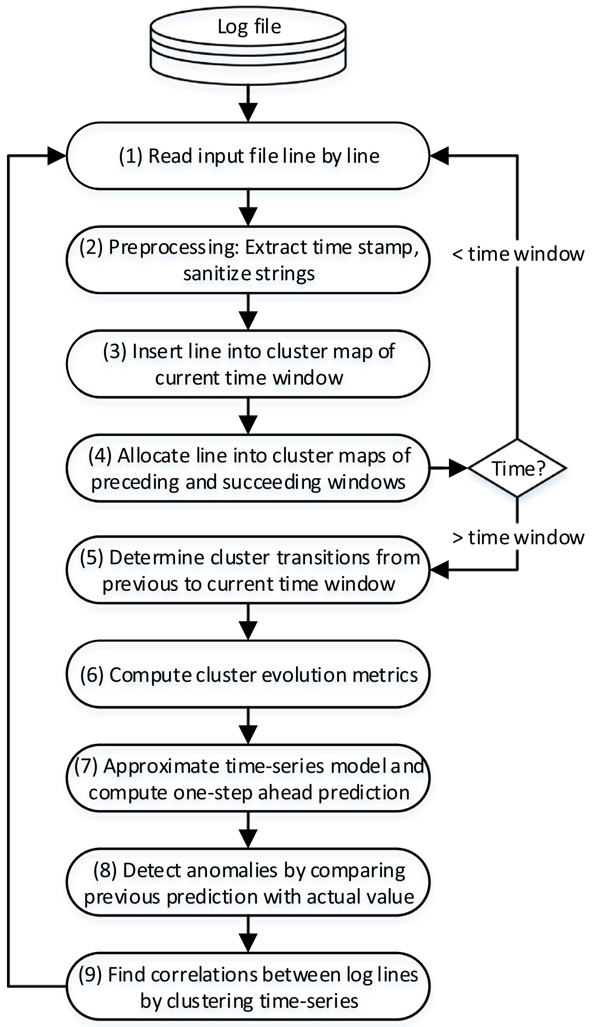

Log file
(1) Read input file line by line
(2) Preprocessing: Extract time stamp, sanitize strings
(3) Insert line into cluster map of current time window
(4) Allocate line into cluster maps of preceding and succeeding windows
(5) Determine cluster transitions from previous to current time window
(6) Compute cluster evolution metrics
(7) Approximate time-series model and compute one-step ahead prediction
(8) Detect anomalies by comparing previous prediction with actual value
(9) Find correlations between log lines by clustering time-series
Time?
< time window
> time window

Fig. 3 – Flowchart of the anomaly detection procedure. Steps (1)–(4) involve clustering, steps (5) and (6) involve cluster evolution and steps (7)–(9) involve time-series analysis.

(7), these cluster metrics that conveniently form time-series are approximated with appropriate models that take trends and seasonal effects into account. An extrapolation of these models is used to produce a one-step ahead forecast for the metric in the subsequent time window. Step (8) then uses the prediction computed in the preceding time window and detects an anomaly if this value deviates too much from the actual measured metric. Thereby it is guaranteed that changes of the system behavior are recognized as fast as possible, while the algorithm is at the same time able to incorporate these changes already in the following prediction. This ensures adjusting to any new state is possible without any need for manual reconfiguration. Finally, step (9) monitors the correlations between the time-series corresponding to the cluster developments. Similar to before, unexpected changes of these correlations indicate that there has been a change of the system behavior. For efficiency, we suggest to group the time-series in a similar way as explained in Section 4, i.e., groups of correlating time-series are incrementally built in every time step. These groups are monitored over time and any evolving cluster that leaves, swaps or joins one of the correlating groups is considered anomalous.

4. Clustering

This section describes the algorithm for generating the static cluster maps within each time window by incrementally adding log lines (Wurzenberger et al., 2017). This algorithm was selected for several reasons. First, it is designed to process data with high performance and thus able to handle large log files. Second, the incremental approach allows processing data that arrives in streams rather than a fixed size data set, which is an essential property for online data monitoring. Finally, there are no restrictions or assumptions regarding the syntax on the input log data due to the fact that string metrics are employed rather than parsers or templates.

These features are enabled by defining each cluster C through a single cluster representative c. This representative is set as the log line that triggered the generation of the cluster and is thus its first member. Furthermore, all clusters are placed in a cluster map $\mathcal{C}$.

The procedure of the algorithm is as follows: The log lines are read line by line from a log file or a log stream. The strings are then preprocessed in order to avoid that certain artifacts have a negative influence on the quality of the clustering. This includes reducing multiple consecutive spaces, removing the timestamp attached to each log line and replacing non-displayable characters. The first log line that is being read always forms a new cluster with itself as the representative as there is no other cluster that this line could be allocated to. For every other line l that follows, a series of checks is carried out. At first, each representative c of all clusters $C \in \mathcal{C}$ is compared for equality with l. In the case that an identical representative is found, the currently processed log line is immediately allocated to the corresponding cluster and the next line is processed. Otherwise, a set of cluster candidates $\mathcal{C}_l \subseteq \mathcal{C}$ is selected based on the lengths of c and l. A cluster C is added to $\mathcal{C}_l$ if $|c|$ lies within a predefined range of $|l|$, e.g., $\pm 10\%$. We argue that this is an efficient method to exclude dissimilar cluster candidates from further comparisons that are computationally more expensive.

The resulting set of cluster candidates is thinned out using a short word filter that compares the amount of k-mers in $c \in \mathcal{C}_l$ and l. Such filters have frequently been applied for clustering biological sequences. The minimum amount M of matching k-mers that is required for C to remain in $\mathcal{C}_l$ is computed by

$$M = L - k + 1 - (1 - p)kL \tag{1}$$

where L is the length of the shorter line, k is the length of the k-mers and p is the similarity threshold selected within the range [0, 1]. This stage again aims at an efficient reduction of the set of cluster candidates.

Finally, the most similar of the remaining cluster candidates $\mathcal{C}_l$ is determined using a string metric, e.g., the well-known Levenshtein distance. For this, the cluster $C \in \mathcal{C}_l$ that minimizes the distance $d(c, l)$ between its representative c and the currently processed line l is selected. If this distance to the most similar representative c is not larger than a predefined threshold t, then l is allocated to C. Otherwise, the log line forms a new cluster with itself as the representative. In addition to clustering log lines within distinct time windows, this algorithm is used for establishing connections between separate cluster maps in the next section.

5. Cluster evolution

An algorithm for clustering log lines was introduced in the previous section. This algorithm is able to operate on a continuous data stream without any fixed end, i.e, the cluster map keeps expanding. While several static features can be extracted from that procedure, for example, the log line types that are responsible for the largest clusters or outliers that indicate unusual log lines, most features about the dynamic cluster developments are lost. This issue is solved by generating several smaller cluster maps within delimited time windows rather than one single large map that keeps growing. However, retrieving dynamic information about individual clusters of multiple time windows is not trivial. Therefore, cluster evolution that aims at the identification of connections between clusters is applied in order to learn about the cluster developments.

5.1. *Cluster tracking*

Cluster tracking analyzes how each of the clusters $C \in \mathcal{C}$ from one time window relate to each of the clusters $C' \in \mathcal{C}'$ from the succeeding time window in order to establish connections between similar clusters from the respective maps. Thereby, two clusters C and C' should be considered similar if their generation was triggered by the same underlying data source, i.e., their contained log lines are similar. However, comparing each log line from one cluster with each other log line from another cluster is computationally expensive and not feasible for large log files. We therefore reformulate the premise in the following way: Two clusters are considered similar if the majority of the elements contained in C' would have been allocated to cluster C if they had been used for the generation of cluster map $\mathcal{C}$. Similar problems have been solved using the Jaccard coefficient for binary sets which measures the ratio of common data points in order to determine the overlap between two clusters:

$$overlap(C, C') = \frac{|C \cap C'|}{|C \cup C'|} \tag{2}$$

This measure has been used by Greene et al. (2010) who compare the overlap with a threshold $\theta \in [0, 1]$ in order to determine whether the clusters match, i.e., whether it can be assumed that C' originates from C. There exist also alternate forms for computing the overlap which use the maximum (Takaffoli et al., 2011) or the minimum (Greene and Cunningham, 2009) of the two set sizes in the denominator and there is also the possibility to use the Hungarian Method, the Max-Flow approach or a linear programming algorithm in order to find the optimal correspondences between clusters of different time steps (Goldberg et al., 2010). In addition, there are variants for determining whether a transition occurred based on measuring the percentage of change for each cluster (Asur et al., 2009).

It is however problematic to make use of this measure in log file analysis as log lines allocated to two clusters in different time windows cannot be regarded as identical, which makes it difficult to reasonably perform set operations such as the union and intersection. This is due to the fact that log lines are just strings that can only reliably be tracked by their line number and equality (i.e., an identical sequence of characters) and continuously changing IDs or timestamps contained in the lines could easily cause that otherwise similar lines from the sets C and C' are not matched due to a single diverging character. The result of this would be that the intersection of the two sets is incorrectly sparse or even empty due to the lack of identical strings.

In order to overcome these problems regarding set operations on log lines, the following strategy was pursued: First, each log line is only referenced by its unique line number once it is allocated to a cluster. This does not only effectively reduce the amount of required memory, but also ensures that identical lines can be differentiated and mathematical restrictions regarding identical members in sets are fulfilled. Second, the key aspect of this procedure is that the log lines occurring during a certain time window are not only used for creating the cluster map of that time step, but are also allocated to the clusters from the cluster maps preceding and succeeding that map. The two phases are called construction phase and allocation phase respectively. In the construction phase, the cluster maps are generated solely by the log lines that actually occur within that time window. On the other hand, the allocation phase allocates log lines from cluster maps that lie either before or ahead of the currently processed cluster map. For clarification, it should be noted that during the allocation phase the lines do not change the existing clusters and also do not induce the generation of new clusters in these maps. While a line that does not fall into any existing cluster would form a new one during the construction phase, it is simply omitted if it does not fit into any existing cluster of another map.

An illustrative example of this procedure is shown in Fig. 4. We point out that the colorings in this illustration are only used for an easier differentiation between the clusters and their members and that at the start of the tracking procedure, it is not known that cluster $C'_{\triangle}$ originates from cluster $C_{\triangle}$ and that cluster $C'_{\bigcirc}$ originates from $C_{\bigcirc}$. In this simple example, 11 log lines occurring over two time windows are considered. The log lines $\{s_1, s_2, \ldots, s_5\}$ occur during the first time window and form two clusters $C_{\triangle}, C_{\bigcirc} \in \mathcal{C}$. As these are the lines used for the creation of the cluster map in this time window, they are seen as the current members of the respective clusters they belong to. Hence, they are stored in the sets of references $R_{\triangle curr} = \{s_1, s_2, s_3\}$ and $R_{\bigcirc curr} = \{s_4, s_5\}$. Analogously, log lines $\{s_6, s_7 \ldots s_{11}\}$ form the clusters $C'_{\triangle}, C'_{\bigcirc} \in \mathcal{C}'$ in the second time window and thus $R'_{\triangle curr} = \{s_6, s_7, s_8, s_9\}$ and $R'_{\bigcirc curr} = \{s_{10}, s_{11}\}$. At this point, the cluster maps are generated and the required references to the generating lines are stored. Thus, the construction phase is finished.

As previously explained, once the two consecutive cluster mappings are established, the allocation phase clusters the

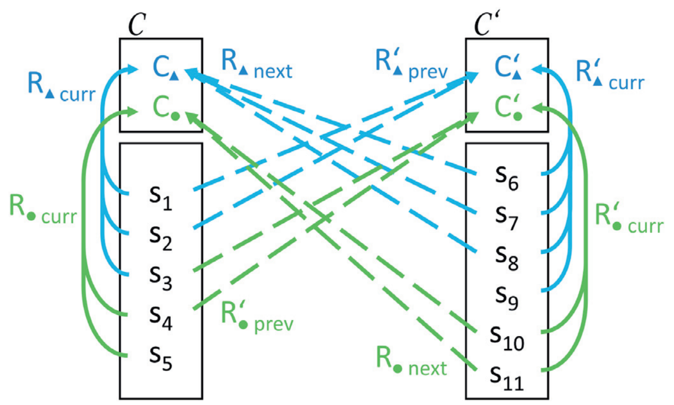


Fig. 4 – Illustrative example how lines are allocated to two different clusters from two consecutive time steps.

lines from each time window into the maps from neighboring time steps. First considering the lines $\{s_1, s_2 \ldots s_5\}$ from the former time window being clustered into the map $\mathcal{C}'$ of the later time step, it can be seen that the allocations lead to the sets of references $R'_{\triangle prev} = \{s_1, s_2\}$ and $R'_{\bigcirc prev} = \{s_3, s_4\}$ stored in the clusters. Analogously, the lines from the later time window were allocated to the clusters from the former time step resulting in the references $R_{\triangle next} = \{s_6, s_7, s_8\}$ and $R_{\bigcirc next} = \{s_{10}, s_{11}\}$. It should be noted that line s_3 was clustered into $C_\triangle$ in the former time step but into $C'_\bigcirc$ in the later time step. A reason for this could be that s_3 has the necessary characteristics to fit into both of the clusters, but due to small deviations in cluster representatives in both time steps the line was not allocated into cluster $C'_\triangle$ unlike the lines s_1 and s_2. This could also be the reason why both s_5 and s_9 were not allocated to any cluster from the neighboring maps.

Furthermore, it should be clear that there is an arbitrarily large number of time steps following and that lines have to be clustered accordingly. For example, assuming that there would be a third time step with a cluster map $\mathcal{C}''$ and its clusters $C''_\triangle$ and $C''_\bigcirc$, their line allocations would be stored in the references $R''_{\triangle curr}$ and $R''_{\bigcirc curr}$. Also, the lines of the second time window would additionally have to be clustered in $\mathcal{C}''$ forming $R''_{\triangle prev}$ and $R''_{\bigcirc prev}$. These connections are not displayed in the figure for simplicity. Finally, the lines of the third time window have to be clustered in $\mathcal{C}'$ forming $R'_{\triangle next}$ and $R'_{\bigcirc next}$. At the end of the allocation phase, references to all the lines from neighboring time windows are stored in each cluster map.

$$overlap(C^1, C^2 \ldots C^N) = \frac{\sum_{j=1}^{N-1} \sum_{i=1}^{N-j} \left| \left(R^i_{curr} \cap R^{i+j}_{prev,j}\right) \cup \left(R^i_{next,j} \cap R^{i+j}_{curr}\right) \right|}{\sum_{j=1}^{N-1} \sum_{i=1}^{N-j} \left| R^{i+j}_{curr} \cup R^{i+j}_{prev,j} \cup R^i_{next,j} \cup R^i_{curr} \right|} \tag{3}$$

Using this kind of cluster allocations, the rule for finding matches between clusters stated in Eq. (2) is adapted to fit the purpose of log line clustering. Using the line references as explained before, the following formula computes the overlap between any two clusters C and C′ of two neighboring maps:

$$overlap(C, C') = \frac{\left| \left(R'_{curr} \cup R'_{prev}\right) \cap \left(R_{next} \cup R_{curr}\right) \right|}{\left| R'_{curr} \cup R'_{prev} \cup R_{next} \cup R_{curr} \right|} \tag{4}$$

Since $R'_{curr} \cap R_{next} = \emptyset$ and $R'_{prev} \cap R_{next} = \emptyset$, this is equivalent to

$$overlap(C, C') = \frac{\left| \left(R_{curr} \cap R'_{prev}\right) \cup \left(R_{next} \cap R'_{curr}\right) \right|}{\left| R'_{curr} \cup R'_{prev} \cup R_{next} \cup R_{curr} \right|} \tag{5}$$

This representation also shows more clearly that the sets R_{curr} and R'_{prev} both contain log lines that were used in the former time step which was also used to create the cluster map $\mathcal{C}$, while both R_{next} and R'_{curr} contain log lines from cluster map $\mathcal{C}'$, thus showing that the intersections are applied reasonably. Dividing the union of these two intersected sets by the union of all sets means that the resulting value is in the interval [0, 1], with 1 indicating a perfect match (i.e., all lines that were clustered into C were also clustered into C′ and vice versa) and 0 indicating a total mismatch.

Using this formula, the overlaps between clusters of the example from Fig. 4 can be computed. For example, the overlap between clusters $C_\triangle$ and $C'_\triangle$ is

$$overlap(C_\triangle, C'_\triangle) = \frac{|\{s_1, s_2, s_6, s_7, s_8\}|}{|\{s_1, s_2, s_3, s_6, s_7, s_8, s_9\}|} \approx 0.714 \tag{6}$$

A more sophisticated clustering model that not only allocates the log lines from a certain time window into the cluster maps of its directly neighboring time windows but also into the ones following after that is able to compute an aggregated overlap over multiple time windows. This means that the overlap from a specific cluster, say $C^1 \in \mathcal{C}^1$, through another cluster $C^2 \in \mathcal{C}^2$ to a third cluster $C^3 \in \mathcal{C}^3$ is computed by not only incorporating the already used references R^1_{next} and R^2_{prev} between C^1 and C^2 as well as R^2_{next} and R^3_{prev} between C^2 and C^3, but also the references between C^1 and C^3. These references are called $R^1_{next,2}$ and $R^3_{prev,2}$, where the additional subscript 2 indicates the distance between the two cluster maps, i.e., cluster map $\mathcal{C}^2$ was skipped. Following this terminology, the references between two directly neighboring cluster maps are called $R^1_{next,1}$, $R^2_{prev,1}$, etc. Analogously, the references between clusters C^i and C^{i+m} that are m steps apart are called $R^i_{next,m}$ and $R^{i+m}_{prev,m}$. The overlap between a sequence of N clusters $C^1, C^2 \ldots C^N$ is computed as follows: Eq. (3). For simplicity, the simple overlap metric is used in the remainder of this paper and the additional index specifying the distance between the cluster maps is omitted.

However, clusters do not necessarily have to have exactly one predecessor and one successor, but can be the product of multiple clusters that merged together or be a part of a larger cluster that split up. In the following, a method for the identification of such transformations is given.

5.2. *Cluster transitions*

The overlap metric proposed in the previous section only sufficiently tracks clusters in simple settings where all clusters are very different from each other, thus yielding high overlaps only with one other cluster. However, the compositions of clusters are frequently subject to change and similar clusters generated in the same time windows may be the result of splits or may result in a merge. Spiliopoulou et al. (2006) propose rules for detecting such advanced transitions using a non-symmetric overlap measure. However, our overlap metric from Eq. (4) is symmetric due to the fact that it considers clusters from the preceding as well as the succeeding cluster map for the calculation. This causes that the sum of all overlaps between a specific cluster and all clusters from the preceding time window as well as the sum of all overlaps between a specific cluster and all clusters from the succeeding time window never exceeds 1. In the following, we use θ as a minimum threshold for the overlap and θ_{part} as a minimum threshold for partial overlaps occurring during splits or merges, where in general $\theta_{part} < \theta$ since partial overlaps yield smaller overlap scores. We adjusted the rules to fit our overlap metric and differentiate between the following transitions:

1. *Survival*: A cluster C survives and transforms into C' if $overlap(C, C') > \theta$ and there exists no other cluster $C_i \in \mathcal{C}$ or $C'_i \in \mathcal{C}'$ so that $overlap(C_i, C') > \theta_{part}$ or $overlap(C, C'_i) > \theta_{part}$.
2. *Split*: A cluster C splits into the parts $C'_1, C'_2 \ldots C'_p$ if all individual parts share a minimum amount of similarity with the original cluster, i.e., $overlap(C, C'_j) > \theta_{part}, \forall j$, and the union of all parts matches the original cluster, i.e., $overlap(C, \bigcup C'_j) > \theta$. There must not exist any other cluster that yields an overlap larger than θ_{part} with any of the clusters involved.
3. *Absorption*: The group of clusters $C_1, C_2 \ldots C_p$ merge into cluster C' if all individual parts share a minimum amount of similarity with the resulting cluster, i.e., $overlap(C_j, C') > \theta_{part}$, $\forall j$, and the union of all parts matches the resulting cluster, i.e., $overlap(\bigcup C_j, C') > \theta$. Again, there must not exist any other cluster that yields an overlap larger than θ_{part} with any of the clusters involved.
4. *Disappearance*: A cluster C disappears if there exists no $C'_i \in \mathcal{C}'$ so that $overlap(C, C'_i) > \theta_{part}$.
5. *Emergence*: A cluster C' emerges if there exists no $C_i \in \mathcal{C}$ so that $overlap(C_i, C') > \theta_{part}$.

The procedure for identifying such cluster transitions is shown in pseudo-code in Algorithm 1 . Line 1 initializes the empty list of all transitions represented as pairs of clusters connected over two time steps. Line 2 initializes an array that holds a list of all predecessor candidates referenced by a cluster from the later time step, i.e., C'. Line 3 initializes another array that holds the summed overlaps for those predecessor candidates. Analogously, Lines 5–6 initialize those arrays for successor candidates that are referenced by a cluster from the former time step, i.e., C. If the overlap between any combination of clusters from different time steps computed in Line 8 exceeds θ_{part} in Line 9, the arrays are updated with this potential connection between the currently processed clusters. In Line 16 it is checked whether the accumulated overlap is larger than θ and only then the connection between a cluster from the former time window and all its successors is added to the list of transitions. Analogously, Line 21 checks the accumulated overlap between a cluster from the later time step and all its predecessors and updates the list of transitions accordingly.

Algorithm 1: Determining external cluster transitions between two time steps.

```
Data: cluster maps $\mathcal{C}, \mathcal{C}'$
Result: list of transitions between pairs of clusters
1  transitions = List();
2  predecessorsCandidates = [List()];
3  predecessorsOverlaps = [ ];
4  for $C \in \mathcal{C}$ do
5      successorsCandidates = List();
6      successorsOverlap = 0.0;
7      for $C' \in \mathcal{C}'$ do
8          overlap = computeOverlap($C, C'$);
9          if $overlap > \theta_{part}$ then
10             successorsCandidates += $C'$;
11             successorsOverlap += overlap;
12             predecessorsCandidates[$C'$] += $C$;
13             predecessorsOverlaps[$C'$] += overlap;
14         end
15     end
16     if $successorsOverlap > \theta$ then
17         transitions += {$C$, successorsCandidates};
18     end
19 end
20 for $C' \in \mathcal{C}'$ do
21     if $predecessorsOverlaps[C'] > \theta$ then
22         transitions += {predecessorsCandidates[$C'$], $C'$};
23     end
24 end
```

In addition to these external transitions, any cluster may be affected by internal transitions regarding one of the following properties:

1. *Size*: The cluster grows in size if C' contains more data points than C, shrinks if C' contains less data points than C and does not change in size otherwise. The size of a cluster C is denoted as $|C|$.
2. *Compactness*: With σ denoting the standard deviation of the distance of the cluster members to the representative of cluster C, the cluster becomes more compact if $\sigma' < \sigma$, becomes more diffuse if $\sigma' > \sigma$ and does not change in compactness otherwise.
3. *Location*: In general, cluster coordinates are used to determine whether a cluster changed its position. In our case however, distances can only be computed relative to other clusters rather than in absolute values.

4. *Skewness*: The skewness γ measures the asymmetry between the cluster members and the cluster representative. The skewness of cluster C decreases if $\gamma' < \gamma$, increases if $\gamma' > \gamma$ and remains constant otherwise.

When tracking a cluster through multiple cluster maps, it might be of interest to assign a continuous identifier to the evolving cluster in order to easily retrieve the cluster properties from every time window. This identifier is required to be robust to changes in cluster structure due to external and internal transitions. While internal transitions pose less of a problem as the cluster itself remains the same, especially splits and merges make it difficult to allocate such an identifier to the clusters as there should not exist more than one cluster evolution with the same identifier. We suggest several possibilities: First, larger cluster sizes may indicate more important clusters, while smaller clusters often contain outliers. Therefore, new identifiers could be created for clusters that have just emerged or for smaller clusters that separate themselves from existing clusters, while their larger sibling clusters obtain the identifier from their common predecessor. Similarly, the cluster resulting from a merge will retain the identifier from the largest of its predecessors. This makes sure that most of the clusters holding high number of members are tracked successfully, while smaller clusters do not interfere with their developments. Another possibility is the focus on the amount of time windows that the preceding cluster was already tracked, i.e., its time of existence. It is reasonable to assume that clusters that have already been existing for a longer amount of time are more stable and therefore a better representation of the system. Furthermore, building upon longer existing clusters results in longer time-series that are better fitted for anomaly detection. Finally, the achieved overlap is another appropriate choice as a higher overlap suggests that the most similar clusters are connected. This means that clusters always retain the identifier from the predecessor with the highest overlap and pass the identifier to the successor with the highest overlap. Moreover, a weighted combination of some or all of the previously mentioned metrics could be used to determine the rules for tracking individual clusters.

5.3. *Evolution metrics*

It is possible to perform anomaly detection on simple cluster features such as the size. However, the cluster size alone does not always give a complete view about the ongoings of the clusters. For example, a cluster that is the result of a merge does not necessarily change in size, but the fact that a transition is taking place may still be a sign of an anomaly. On the other hand, omitting any information about the advanced transitions may hide that a change of size is caused by a merge.

Therefore, measures that represent features of individual clusters, combinations of clusters or the whole cluster map are required. First, metrics that are computed from individual clusters and single time windows are considered. These include the size of the cluster or the average distance and variance of all cluster members to the cluster representative. More sophisticated analyzes could be carried out regarding the distribution of the members. Other metrics are based on transitions, i.e., they measure the change of a tracked cluster from one time window to another. Toyoda and Kitsuregawa (2003) state several examples of such metrics. However, they observe the same elements over multiple time windows which is not possible for non-recurrent log lines. Thus, the advantages of the proposed bidirectional clustering method are utilized similar to the computation of the overlap metric. In the following, we state a selection of metrics that take advantage of our clustering model:

1. *Growth rate*: Measures the absolute difference between the member sizes of two consecutive time steps. For a reasonable interpretation of the value the time windows should be of equal size.

$$\rho_{grow} = \left|R'_{curr}\right| - |R_{curr}| \tag{7}$$

2. *Change rate*: Measures the relative difference between the cluster allocations of the lines from the former time step with respect to the total number of lines that were processed in the corresponding time window. Other than the growth rate, this metric only takes lines from the former time window into account but could also be computed for the latter time window.

$$\rho_{change} = \frac{\left|R'_{prev}\right| - |R_{curr}|}{\left|\bigcup_{C_i \in \mathcal{C}} R_{curr}\right|} \tag{8}$$

3. *Stability rate*: Measures the fraction of appeared, disappeared, merged and split members between two consecutive time steps. Note that 0 indicates that all log lines that were allocated to this cluster in one time step were also allocated to this cluster in the other time step and thus the cluster is considered as stable, while 1 indicates that none of the allocations coincided with the other time step and thus the cluster is instable.

$$\rho_{stab} = \frac{\left|R'_{prev}\right| + |R_{curr}| - 2 \cdot \left|R'_{prev} \cap R_{curr}\right|}{\left|R'_{prev}\right| + |R_{curr}|} \tag{9}$$

4. *Novelty rate*: Measures the fraction of newly appeared members between two consecutive time steps. Different than the growth rate, the novelty rate only considers log lines that were allocated to C' but not C. The related disappearance rate is computed by switching R'_{prev} with R_{curr}.

$$\rho_{novelty} = \frac{\left|R'_{prev} \setminus R_{curr}\right|}{\left|R'_{prev}\right|} \tag{10}$$

5. *Split rate*: Measures the fraction of members that were split from C between two consecutive time steps. The related merge rate is computed by switching R'_{prev} with R_{curr} and replacing $\mathcal{C}'$ with $\mathcal{C}$.

$$\rho_{split} = \frac{\left|\left(R_{curr} \cap \bigcup_{C'_i \in \mathcal{C}'} R'_{prev}\right) \setminus R'_{prev}\right|}{|R_{curr}|} \tag{11}$$

Except for the growth rate, switching R_{curr} with R_{next} and also R'_{prev} with R'_{curr} yields metrics based on the log lines appearing in the latter time window. Moreover, the stated equations only take exactly two clusters into account. However, it may be desirable to treat a cluster that originates from multiple clusters due to absorption or transforms into multiple clusters due to splitting in a different way. For example, the union of the cluster fragments or the average of multiple rates computed for each individual cluster fragment could be considered.

6. Time-series analysis

The metrics mentioned in the previous Section are computed once for every time window. Due to the measurements in these regular intervals, time-series are generated that conveniently allow the application of time-series models. We decided to employ the well-known ARIMA models as they take trends and periodical effects into account and support forecasting. Moreover, it is easy to fit an autoregressive integrated moving-average (ARIMA) model to any given time-series, e.g, through a search over different model parameters that minimize a predefined quality criterion.

6.1. Forecasting

After finding appropriate model parameters that reasonably approximate the time-series $y_1, y_2 \dots y_N$ of length N, the properties of this model can be used to extrapolate over the last recorded time step and thus create a forecast for upcoming values (Cryer and Chan, 2008). This procedure is known as one-step-ahead prediction as it aims at approximating an unknown future value $\hat{y}_{t+1}$ that follows directly after the most recent point y_t. There is also the possibility to apply this procedure recursively in order to predict for longer horizons, but since we fit a new model in every time step there is no need to produce forecasts for longer time spans. Furthermore, it is not necessary to include all historic data in the prediction, since the more recent values have more influence on the current prediction. We therefore argue that only a predefined amount of the most recent values should be kept in memory at any time.

In addition to the predictions, we require a measure for the spread that states the trust in the forecasts. For random variables, this is usually solved with confidence intervals that contain the unobservable true parameters with a specified probability. Contrary to that, prediction intervals are associated with an unknown random variable. The prediction interval that is used in the following therefore contains the actual future value with the specified probability (Hyndman, 2013).

Checking whether a future value lies within the prediction interval is an effective method for detecting contextual anomalies, i.e., data points that are anomalous with respect to the most recent values. Using one-step ahead prediction, this means that the most recent measured value is compared with the prediction interval generated one step before. In mathematical terms, the currently processed value y_t is identified as an anomaly if

$$y_t \notin \left[\hat{y}_t - \mathcal{Z}_{1-\frac{\alpha}{2}} s_e, \hat{y}_t + \mathcal{Z}_{1-\frac{\alpha}{2}} s_e\right] \tag{12}$$

where $\hat{y}_t$ is the prediction, $\mathcal{Z}_{1-\frac{\alpha}{2}}$ is the quantile $1-\frac{\alpha}{2}$ of the standard normal distribution and s_e is the standard deviation of the error, $s_e = \sqrt{\frac{1}{n-1}\sum(y_t - \bar{y}_t)^2}$.

6.2. Correlation

Correlation is frequently used to measure the relatedness of variables that do not necessarily need to be in a causal relationship. In general, positive correlation means that a change of one variable in a certain direction indicates that there is also a change of a correlated variable in the same direction. This principle can also be applied to time-series rather than variables and is represented by the cross-correlation function (*CCF*) (Cryer and Chan, 2008). The *CCF* measures whether the two time series follow a common pattern, i.e., whether the slopes from one step to another correspond in each series, show an inverted behavior or no relationship at all. Some time-series may also require to be shifted by a certain lag in order to correlate. The *CCF* can be computed for the two time-series $y_1, y_2 \dots y_N$ and $z_1, z_2 \dots z_N$ and any lag k by

$$CCF_k = \begin{cases} \dfrac{\sum_{t=k+1}^{N}(y_t - \bar{y}) \cdot (z_{t-k} - \bar{z})}{\sqrt{\sum_{t=1}^{N}(y_t - \bar{y})^2}\sqrt{\sum_{t=1}^{N}(z_t - \bar{z})^2}} & \text{if } k \geq 0 \\ \dfrac{\sum_{t=1}^{N+k}(y_t - \bar{y}) \cdot (z_{t-k} - \bar{z})}{\sqrt{\sum_{t=1}^{N}(y_t - \bar{y})^2}\sqrt{\sum_{t=1}^{N}(z_t - \bar{z})^2}} & \text{if } k < 0 \end{cases} \tag{13}$$

The occurrences and frequencies of some of the log line types correlate over time which means that the time-series of the retrieved cluster sizes also correlate with each other. The reason for such correlations can be diverse. For example, certain log lines may always be generated in pairs because their components are linked with each other. On the other hand, scheduled programs may happen to run synchronous by coincidence. In any way, the resulting CCF_k of related time-series should therefore be constant over time. Deviations of long-term trends are indicators of system behavior changes that should be detected as anomalies. For efficiency, we suggest to group the time-series in the same way that the clustering of the log lines was accomplished, i.e., groups of correlating time-series are incrementally built in every time step. These groups are monitored over time and any time-series that leaves, swaps or joins one of the correlating groups is considered anomalous.

The correlation analysis is the final step of the concept outlined in Section 3. Using the proposed procedure, anomalies can be detected in every evolving cluster. However, it may be desirable to aggregate the detected anomalies in every time step. Such an aggregation strategy is pursued in the following section.

6.3. Aggregated detection

The previously explained procedure specifically aims at detecting anomalies for a specific cluster, i.e., each detected anomaly is associated with exactly one cluster. This clearly has some advantages, e.g., the cluster representative or the cluster members immediately give information about the exact type of log line that is affected by the anomaly and it may therefore be easier to trace back the source of the problem.

However, the enormous amounts of clusters in combination with the probability-based approach of the prediction limits naturally causes a rather high number of false alarms that are raised in each time step. For example, if a prediction interval that contains the actual value with 99% probability is computed for 100 clusters in each time step, 1 false alarm is raised per time step on average. In practical applications it is therefore a tedious task to react to every single alarm that is raised and there is a therefore need for a more robust measure.

An intuitive way to solve this problem is to aggregate the anomalies that occur in each time step. On average, randomly occurring anomalies caused by natural fluctuations and noise should occur uniformly distributed over time and are unlikely to collectively occur in multiple clusters at a single point in time. Given that an actual anomaly usually affects more than 1 clusters, counting the number of clusters that report anomalies is therefore a reasonable start. However, this does not incorporate that an anomaly that lies far outside of the prediction interval should be considered as more anomalous than an anomaly that just barely exceeds the upper or lower limit. An aggregated anomaly score should therefore yield larger values that indicate a more severe anomalous state if more clusters report anomalies and more of the reported deviations have a large magnitude.

Furthermore, not all clusters should be equally weighted when considering the anomalies that are detected in their developments. A cluster that has only recently emerged is likely to report more false alarms due to the fact that too few historic values are available to properly compute the prediction interval. On the other hand, clusters that have been existing for a large number of time steps are more likely to exhibit stabilized features and are therefore more trustworthy. Contributions to the anomaly score of a time step should hence be weighted according to the respective durations that a cluster has already been existing.

Before introducing such an aggregated anomaly score, a value s_t that mirrors anomalous y_t values that fall below the lower limit of the prediction interval to the upper side is defined for convenience. With the upper prediction limit $u_t = \hat{y}_t + \mathcal{Z}_{1-\frac{\alpha}{2}}\sqrt{Var(e)}$ and the lower prediction limit $l_t = \hat{y}_t - \mathcal{Z}_{1-\frac{\alpha}{2}}\sqrt{Var(e)}$, the mirrored value is defined as

$$s_t = \begin{cases} y_t & \text{if } y_t > u_t \\ 2\hat{y}_t - y_t & \text{if } y_t < l_t \end{cases} \tag{14}$$

Note that the first case corresponds to y_t lying above the prediction limit, meaning that no action is necessary. The second case corresponds to y_t lying below the prediction limit causing that the point needs to be mirrored around the predicted value $\hat{y}_t$ which is always positioned in the center of the prediction interval. Therefore, the distance between y_t and the closest prediction limit will remain the same after mirroring. Furthermore, the set of clusters containing an anomaly at time step t is defined as

$$\mathcal{C}_A^t = \{C \in \mathcal{C}^t : y_t > u_t \vee y_t < l_t\} \tag{15}$$

With these definitions and the duration τ_t that measures how many time steps a cluster has already been existing, the anomaly score a_t at time step t is defined as

$$a_t = \begin{cases} 0 & \text{if } \mathcal{C}_A^t = \emptyset \\ 1 - \frac{\sum_{C\in\mathcal{C}_A^t}(u_t \cdot log(\tau_t))}{\left|\mathcal{C}_A^t\right| \cdot \sum_{C\in\mathcal{C}_A^t}(s_t \cdot log(\tau_t))} & \text{otherwise} \end{cases} \tag{16}$$

Both u_t and s_t are multiplied with the same $log(\tau_t)$ in order to give clusters that have been existing for a longer time more weight. The logarithm was used to dampen this effect.

Note that u_t in the numerator defines the upper limit of the prediction interval and the variable s_t in the denominator represents the actual value. It is known that $s_t > u_t$ due to the fact that only clusters that contain an anomaly at time step t are considered in the sum and actual values $y_t < l_t$ have been mirrored to the upper side. As both terms are weighted equally, the denominator must always be larger than the numerator and therefore the division is guaranteed to be smaller than 1. Larger deviations from the expected value, i.e., a higher value for s_t, hence cause that the division yields values closer to 0.

Furthermore, including the term $|\mathcal{C}_A^t|$ in the denominator accounts for the impact of more clusters reporting anomalies. Again, a higher amount of clusters reporting anomalies draws the resulting value closer to 0. Finally, the result is subtracted from 1 in order to have anomaly scores close to 0 indicating normal behavior while anomaly scores close to 1 indicate anomalous behavior. In practice, an alarm should be raised if the anomaly score exceeds some predefined threshold.

7. Evaluation

As outlined in the Section 1, unsupervised methods are able to detect anomalies on unlabeled data. While this is a beneficial setting in practical applications where labeled data is barely available, a proper evaluation cannot be carried out on a largely unknown data set. The reason for this is that there is no way to tell whether detected anomalies actually correspond to real anomalies that occurred in the system and whether most of the anomalies in the data have actually been detected. While it would be easy to perform the evaluation on synthetic log data, one could criticize that this kind of data does not resemble practically relevant log data and is therefore not appropriate for a realistic evaluation. As a compromise, the evaluation in this article is carried out on a semi-synthetically created log file that only contains specific anomalies that occur at known points in time. This combines the advantages of the real world data by incorporating sufficient complexity and the advantages of synthetic data by enabling the creation of a ground truth table, i.e., a complete set of anomalous log line types that are known to appear at specific points in time.

7.1. *Log data*

The generation of the log data was carried out by adapting the approach introduced by Skopik et al. (2014). The setting consists of a MANTIS Bug Tracker System[1] deployed on an Apache web server. A variable amount of virtual users simulate real user behavior by navigating on the website. The users perform actions just as real users would do, including reporting, assigning and deleting issues as well as clicking on entries from the task menu and regularly logging in and out. Log lines generated by their behavior are highly complex due to random numbers determining which actions are taken and what kind of selections are made in each step. Clicking on a certain button therefore does not always generate an identical set of log lines, especially because they frequently contain the current date or time, IDs as well as random selections, numbers and strings. Clustering of the resulting log lines thus requires the fuzzy matching approach that was explained in Section 4 and could hardly be accomplished using parsers or templates. Furthermore, some types of log lines (e.g., "Init DB" and "Quit") are produced for every single SQL query, while others only occur when a special action is performed. This is an important characteristic as it implies that the caused anomalies may be of different magnitude in each cluster. It should also be noted that any action usually leads to the generation multiple log lines and therefore anomalies may manifest themselves in multiple clusters.

Logs are recorded from three components: The Web Server, the SQL database and the reverse proxy. The logs therefore contain the accessed URLs, user-specific data such as MAC addresses as well as executed SQL queries.

With this setup, an illustrative attack scenario is introduced. The scenario takes place over the course of 96 h (4 days) and was simulated in real-time. Five virtual users are involved in the creation of the logs. Three of them continuously produce log lines corresponding to normal behavior, i.e., the probabilities of performing certain actions are always constant. Another user simulates an automatized software or program that operates only in the first 30 min in every hour, resulting in a periodic behavior of the affected clusters. The behavior of these four users is considered to be normal behavior that is free of anomalies. For this scenario, the final user poses an intruder who gained unauthorized access to the system after a social engineering attack. The frequencies of performed actions by this user do not cohere to the overall behavior of the others. Over the course of the simulation, the attacker performs the following anomalous actions:

1. *Missing periodic event*: After 17 hrs, the intruder blocks the automatized program for 1 h from performing the scheduled event. The log lines corresponding to the planned actions in this time window are therefore absent from the log file. Afterwards, the program continues to work as usual.
2. *Sudden frequency peak*: After 35 hrs, the attacker clicks on a specific button for a duration of 10 min. The probability of this event is higher compared to the other users, thus the recorded frequency of the corresponding log lines increases.
3. *Long-term frequency increase*: After 53 hrs, the intruder clicks on another button for the following 8 hrs. Again there is a higher probability of this event, resulting in a plateau in the recorded frequency of the corresponding log lines.
4. *Gradual frequency increase*: After 79 hrs, the attacker clicks on a third button until the end of the simulation. It is assumed for this case that the attacker knows about the installed anomaly detection system and therefore tries to outsmart the algorithm by avoiding rapid changes in frequency that may trigger alarms, while at the same time the learning effect of the algorithm adapts to the malicious behavior. After some time, the attacker is able to further increase the clicking frequency. By continuing this pattern for a sufficient duration, the attacker should be able to inject arbitrary large frequency changes. This attack was purposely added to point out deficiencies that arise from self-learning anomaly detection systems.

[1] MANTIS Bug Tracker available at https://www.mantisbt.org/, accessed 08-November-2017.

Fig. 5 shows these attacks on a timeline. Large gaps of several hours were intentionally left between the injections in order to ensure that previous attacks do not affect the likelihood of a future attack being detected. In total, the generated log file consists of around 4 million log lines. The average length of the log lines is around 246 characters in the raw form and around 218 characters after removing consecutive white spaces during the preprocessing stage. More than 99.7% of the log lines have a length below 600 characters.

7.2. *Evaluation environment*

The log data was generated on a general purpose workstation, with an Intel Xeon CPU E5-1620 v2 at 3.70 GHz 8 cores and 16 GB RAM, running Ubuntu 16.04 LTS operating system. The workstation runs virtual servers for an Apache Web server hosting the MANTIS Bug Tracker System, a MySQL database and a reverse proxy. The log messages of these systems are aggregated using syslog.

The detection algorithm was implemented in Java version 1.8.0.141 and runs on a 64-bit Windows 7 machine, with an Intel i7-3770 CPU at 3.4 GHz and 8 GB RAM.

7.3. *Results*

Evaluation performance typically depends on parameter finetuning. Instead of searching for optimal parameter values, several settings are tested and compared in order to reveal the influences of each parameter. Insights gained by such experiments are expected to generalize also on other data sets and aid the identification of appropriate parameter ranges in practical applications. Important parameters and their default values are as follows:

- Similarity threshold t: The threshold used for the incremental generation of the static cluster maps within each time window. A higher threshold means that log lines must be more similar in order to be grouped within the same cluster. This also means that a higher threshold usually relates to a higher total amount of clusters. Unless otherwise stated, $t = 0.9$.
- Overlap thresholds θ and θ_{part}: The thresholds used within the transition detection algorithm. A higher threshold θ means that clusters from different cluster maps require a

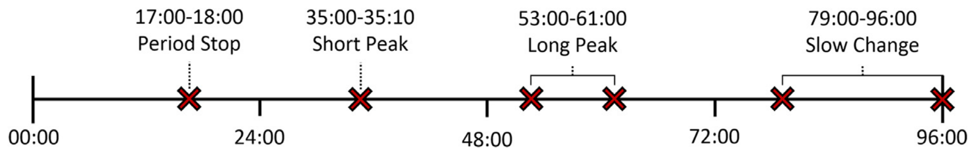


Fig. 5 – Timeline of the attacks.

higher overlap in order to be connected. Furthermore, θ_{part} specifies the minimum overlap that is required for clusters that contribute to a transition, i.e., to be part of a merge or split. Unless otherwise stated, $\theta = 0.7$ and $\theta_{part} = 0.2$.

- Time window size tw: The cluster maps are generated within each time window. A larger time window size therefore means that more log lines are used for the creation of each cluster map. Unless otherwise stated, $tw = 15$ min.
- Prediction level α: The prediction level is used in the quantile $1 - \frac{\alpha}{2}$ of the standard normal distribution $\mathcal{Z}_{1-\frac{\alpha}{2}}$ when computing the prediction intervals. A higher prediction level leads to a smaller intervals and therefore increases the amount of detected anomalies. Unless otherwise stated, $\alpha = 0.01$, i.e., $1-\alpha = 99\%$ of the non-anomalous data points should be located within the boundaries.

7.3.1. *Operability*

The introduced clustering model and the anomaly detection mechanism are designed to only focus on dynamic changes that occur over multiple time windows rather than other forms of anomalies that occur only in a single time window. Such other anomalies are for example outliers, i.e., log lines that form their own cluster in the construction phase due to their high dissimilarity to all the other lines and are also not allocated to clusters from other time windows during the allocation phase. Clearly, there is no way to identify any temporal changes from such lines as they simply do not exhibit any dynamic features.

While outliers are an extreme example, also clusters containing more than 1 element and existing for several time steps cannot always be used for detection. Due to the fact that the ARIMA model requires a number of historic data points before the prediction interval is reasonably initialized, only anomalies detected in clusters that have been existing for at least 5 time steps are considered. This is necessary to avoid the relatively high amount of false alarms that occur in the first few time steps and impair the evaluation results.

There may therefore be a concern that only few log lines remain that are eventually contained in the cluster evolution process. Such a situation would indicate a low credibility and could also lead to a poor performance of the algorithm due to the fact that most of the log lines are never considered for the anomaly detection procedure.

It is therefore important to understand the factors that influence the ability of forming permanent and stable clusters that exist for at least the minimum amount of time steps required for a proper anomaly detection. For a given data set, the functioning of the clustering model in combination with the overlap coefficient determines whether clusters are effectively mapped over time. The most relevant parameter is thus the similarity threshold t used in the clustering process. Fig. 6 shows the relative amount of log lines contained in evolving clusters that exist for at least 5 time steps plotted against t. It can clearly be seen that low thresholds ($t \leq 0.5$) cause that only $20\% - 30\%$ of the total amount of log lines end up in evolving clusters while large thresholds (especially $0.8 \leq t \leq 0.9$) achieve a representation of more than 90% of all log lines. There is thus a clear preference towards larger thresholds.

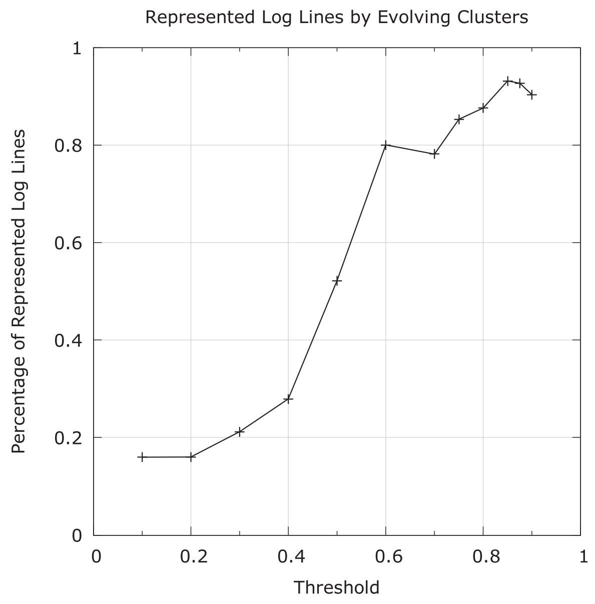


Fig. 6 – Effectiveness of cluster evolution approach evaluated by the relative amount of log lines that are represented by an evolving cluster that exists for at least 5 time steps.

The reason behind this tendency is as follows. Lower values for t lead to fewer clusters in each cluster map as well more diverse types of log lines being grouped into the same clusters. Due to the fact that there is always only one cluster representative responsible for representing all the contained log lines, also largely dissimilar log lines are represented by this initial line as only a small similarity between the strings is required. However, in other time steps it is possible that different cluster representatives are selected for clusters with otherwise similar contents. In other words, low similarity thresholds cause that the cluster representatives are not appropriately representing the log lines allocated to the clusters. This is problematic when it comes to the allocation phase, since the log lines that established a cluster in one time step are therefore likely to be allocated to several clusters from another time step. Thus, only few distinct connections between single clusters can be made, because the minimum cluster similarity thresholds θ and θ_{part} for establishing transitions are hardly

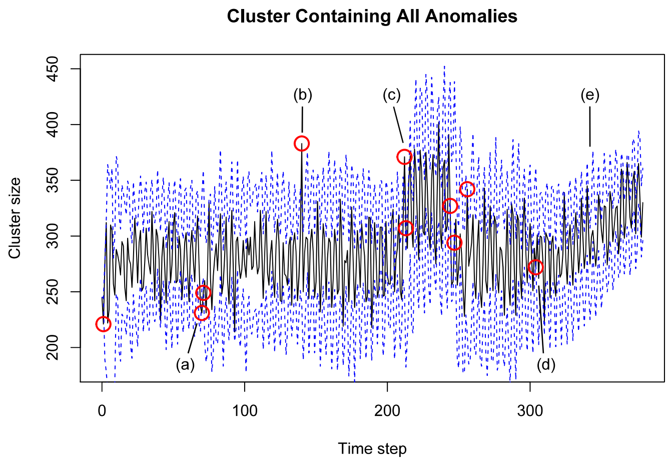


Fig. 7 – Development of cluster corresponding to log line "Init DB". Solid black line: Actual measured cluster size. Dashed blue line: One-step ahead prediction boundaries. Red circles: Detected anomalies.

ever reached. Without transitions, no cluster evolution takes place and hence there remain large clusters in every time step that do not have any correspondences in the preceding or succeeding time windows and are thus unable to contribute to dynamic anomaly detection.

On the other hand, large thresholds lead to the formation of many clusters, with most of them containing highly similar log lines. This corresponds to a finer granularity of clustering. In every time window, the cluster maps consist of clusters with similar representatives and thus log lines from one cluster are correctly allocated to a specific cluster in another time step during the allocation phase. Thereby, distinct connections between clusters are established and the overlap metric successfully creates transitions between the cluster maps, resulting in many evolving clusters.

7.3.2. *Cluster evolution visualizations*

Visualizations of the time-series retrieved by cluster evolution techniques aid the understanding of the anomaly detection mechanism and demonstrate the functionality of the introduced approach. Only taking evolving clusters that were tracked for at least 20 time steps into account, over 300 such clusters were found. In the following, we discuss plots of some interesting clusters evolutions.

There are log lines that appear more frequently than others, e.g., every set of SQL queries belonging to a certain action always start with a log line stating "Init DB". All of the injected attacks involve the creation of SQL queries, therefore the cluster corresponding to this line is expected to display the effects of all attacks. Fig. 7 shows the development of the size of this cluster over time. Note that each time step covers a 15 minute period of occurring log lines, thus the total amount of 384 time steps corresponds to a time span of 96 h. The actual measured cluster size (solid black line) is approximated in every step in order to predict the boundaries (dashed blue lines) for the following step. Whenever the actual size in the next step falls outside of the tube that is formed around the curve, an anomaly is detected and marked with a red circle. The figure shows the following features:

(a) A correctly detected anomaly, i.e., a true positive. This anomaly is corresponding to the missing periodic event attack. The figure shows that the periodic behavior is captured very well throughout the simulation as the position of the prediction interval corresponds to the up-and-down movements of the cluster size. The time-series model learns the correct period in less than 10 time steps and is further able to keep the correct periodicity while adjusting to outliers (b), level shifts (c) and changes in trend (e).
(b) Another correctly detected anomaly corresponding to the sudden increase in frequency. Due to the fact that the duration of this attack is smaller than the length of a time window, only one time step is affected.
(c) Another correctly detected anomaly corresponding to the start of the long-term increase in frequency. Also the decrease of frequency at the end of the plateau is detected correctly. The figure clearly shows that it only takes only few time steps until the time-series model adapts to the new mean value as there are no anomalies detected in between the start and the end of the plateau. This demonstrates the self-learning ability of the time-series model to adapt to changing environments without the need to manually interfere.
(d) An incorrectly detected anomaly, i.e, a false positive.
(e) An undetected anomaly, i.e., a false negative. This anomaly corresponds to the gradual frequency increase. As expected, this anomaly is not detected by the time-series model due to the fact that the frequency change is not rapid enough in any time step so that the actual cluster size would fall outside of the tube.

All other points are therefore correctly undetected, i.e., true negatives. One of the main advantages of this anomaly detection methodology is that many evolving clusters that are specific to certain log line types are retrieved from the log file. Thereby, each cluster development may exhibit some specific characteristics that would remain unnoticed when only considering the log file as a whole. Fig. 8 shows such a cluster that does not have a periodic component, i.e., the log lines contained this cluster are not part of the log lines created by the

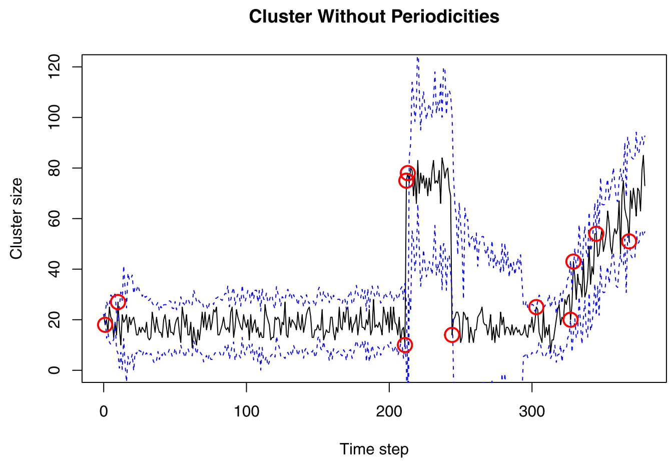


Fig. 8 – Development of a cluster size that corresponds to log lines affected by anomalies regarding long-term frequency increase and the gradual frequency increase.

periodically triggered event. Therefore, the curves do not show regular up-and-down movements. This example shows that the ARIMA model initially requires a number of time steps until stable and appropriately sized prediction intervals are computed. After around 20 time steps, the tube flows around the cluster size with a constant width and no single false positive anomaly is detected in the first 200 time steps. Clearly, neither the missing periodic event anomaly nor the short-term frequency peak anomaly are detected in this cluster size development, because the log lines generated by these events are not contained in this evolving cluster.

It is the specificity of this cluster and the lower amount of contained log lines that make it easier to detect the remaining anomalies. Although Fig. 7 indicated the detected anomalies as well, the magnitude of the change was rather low, i.e., the average cluster size in each time step increased only by 25% from around 280 to 350 when the long-term frequency increase anomaly occurred. The cluster size displayed in Fig. 8 however increases by 400% from around 20 to 80. Furthermore, this plot shows several anomalies detected during the gradual frequency increase anomaly while no anomalies were detected in the previous plot. Especially in the case where a higher prediction level (i.e., a larger thickness of the tube) is used, anomalies possibly remain undetected in larger clusters, but stay visible in smaller clusters. It should now be apparent that it is necessary to consider all cluster size evolutions for anomaly detection in order to ensure that anomalies manifesting themselves only in very specific clusters are detected as well.

This visualization also shows the influence of an anomaly on the following forecasts of the prediction interval. As it can be seen, the interval increases to an unexpectedly large thickness for around 50 time steps after strong deviations from normal behavior occur. This is the result of the large error computed between the anomalous value and the estimated value increasing the range according to Eq. (12). Because of these effects, anomalies occurring within a certain amount of time steps after an attack are more unlikely to be detected since they may fall within the larger intervals despite any large deviations from normal behavior. It would therefore be intuitive to remove the errors generated by anomalies from the computations in order to avoid these adverse effects. It must however be noted that the errors generated by false positives are essential for producing correctly sized prediction intervals in the following time steps. Given that false positives usually outnumber true positives we therefore recommend to keep all the errors for the computations, despite their influence on future predictions. In practical applications however, a reasonable compromise would be to omit the errors from anomalous values that have been confirmed as true positives by a human system administrator.

Fig. 9 shows the development of a cluster size that contains periodically occurring log lines, where only the segment of the time-series that contains the missing periodic event around time step 70 is displayed. Due to the fact that these corresponding lines appear very regularly and there are no other log lines causing noise or other fluctuations, the ARIMA model is able to approximate the curve very closely. Even though the anomaly is correctly detected, the ARIMA model does not unlearn the normal behavior of the tube, so that once the curve has returned to its normal behavior, the tube is already in an appropriate shape.

7.4. *Rates*

Quantitative metrics are required for an appropriate comparison of results achieved in different settings. As already mentioned, a ground truth table containing the time steps and samples of log lines that were generated during the respective attacks was assembled. An anomaly is detected by the algorithm at a specific detection time step t_d and for a specific cluster with representative r_d. Anomalies are only counted as true positives (*TP*) if the ground truth table contains an entry with expected time step $t_e \in [t_d - 30min, t_d + 60min]$ and expected log line content r_e so that $s_{Lev}(r_e, r_d) \geq t$, i.e., the similarity must be greater or equal to the threshold that was used for clustering. Detected anomalies that do not fulfill one of these requirements are counted as false positives (*FP*). Entries from the ground truth table that remain undetected are counted as false negatives (*FN*), i.e., actually occurring anomalies that

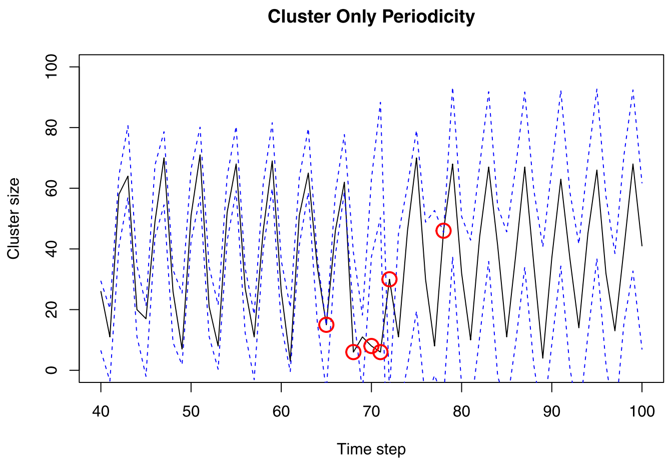


Fig. 9 – Detailed view on the segment where the missing periodic event anomaly occurs.

remained undetected. The amount of true negatives (*TN*) is determined computationally by summing up all the time steps in every cluster that were not detected as anomalies and subtract *FN*. Table 1 gives an overview about the relationships of these values in a confusion matrix.

It is common to compute rates based on these measures (Powers, 2011). The true positive rate (*TPR*) represents the fraction of correctly detected anomalies from the total amount of actually existing anomalies and is computed by

$$TPR = \frac{TP}{TP + FN} \tag{17}$$

Clearly, $TPR \in [0, 1]$ and a high *TPR* is favorable as it implies that many of the actually existing anomalies have been detected. However, the *TPR* neglects the amount of *FP* and is therefore on its own not an appropriate measure for determining the overall quality of the detection.

In order to overcome this problem, the false positive rate (*FPR*) is frequently used in combination with the *TPR*. The *FPR* represents the fraction of incorrectly detected anomalies from the total amount of non-anomalous data points. The rate is computed by

$$FPR = \frac{FP}{FP + TN} \tag{18}$$

and again $FPR \in [0, 1]$. Anomaly detection techniques try to keep the amount of false alarms at a minimum, hence a lower *FPR* is favorable.

7.4.1. ROC analysis

The previously mentioned metrics are now used for creating the familiar Receiver-Operator-Characteristic (ROC) (Powers, 2011), where *TPR* on the y-axis is plotted against *FPR* on the x-axis. Accordingly, classifiers that yield points close to the top-left corner ($TPR = 1$ and $FPR = 0$) of the ROC plot are favorable. We used the prediction level α in order to connect the points yielded from certain parameter settings. A low value for α leads to a large prediction interval and therefore only anomalies with extreme deviations are detected, i.e., the algorithm will miss most of the anomalies but also exhibit a low false alarm rate ($TPR \approx 0$, $FPR \approx 0$). On the other hand, a high value for α leads to a small prediction interval which will lead to almost all data points being detected as anomalies ($TPR \approx 1$, $FPR \approx 1$). In between lies a desirable trade-off value that maximizes *TPR* and minimizes *FPR*. Moreover, we added the first median ($TPR = FPR$) in the ROC plot which shows the performance of a random guesser.

Table 1 – Confusion matrix.

		Actual state	
		Anomalous	Normal
Detected state	Anomalous	TP	FP
	Normal	FN	TN

Fig. 10 shows the ROC analysis for different settings of the similarity threshold *t*. Thresholds smaller than 0.5 were omitted due to their deficiencies discussed in Section 7.3.1. Corresponding to the insights regarding the percentage of contained log lines outlined in that section, $t = 0.85$ and $t = 0.875$ outperform smaller and larger thresholds in the ROC analysis. These results are interpreted as follows: For an appropriate performance, *t* must be large enough to correctly differentiate the occurring log line types while still being small enough to avoid the creation of outliers due to IDs, time stamps or other artifacts in the strings.

Every curve shows the trade-off between a high *TPR* and a low *FPR*. In practice, α should be set so that the resulting point lies at the “bend” of the curve. Since it is usually sufficient to detect an anomaly in at least one cluster, we argue that minimizing *FPR* should be of primary focus. For $t = 0.875$ a practically reasonable α would thus be 0.001 as it achieves $TPR = 0.618$ and $FPR = 0.007$.

The time window size *tw* is more difficult to choose since good values rely on assumptions about the data. A small time window size corresponds to a fine granularity, i.e., changes that occur in short periods are more distinctly present. Accordingly, a small value for *tw* would be beneficial for detecting attack causing a short-term frequency increase. On the other hand, large window sizes have the advantage of producing less

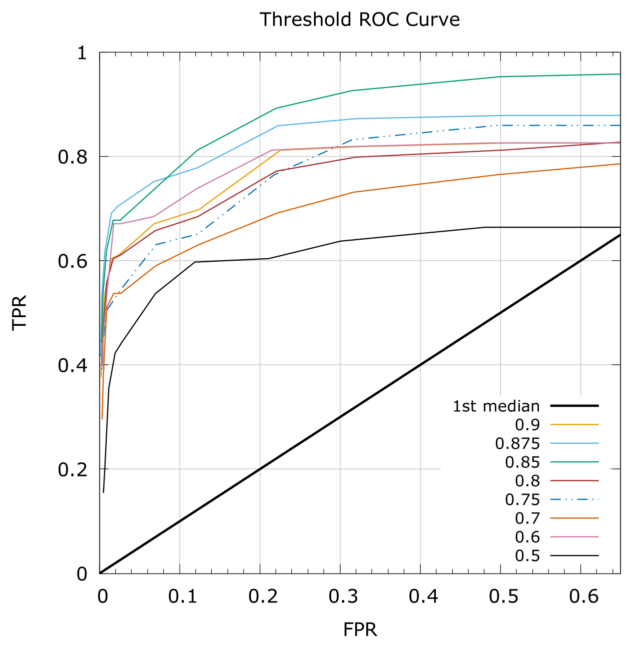


Fig. 10 – ROC curves showing anomaly detection performance for different similarity thresholds.

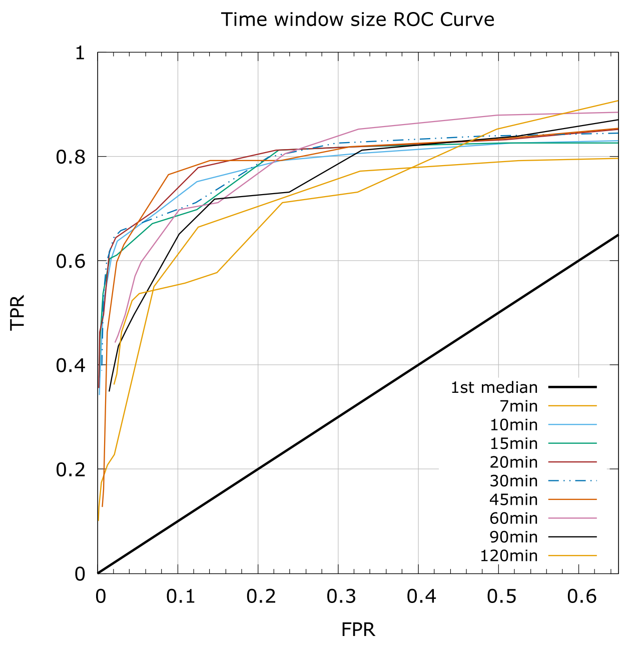


Fig. 11 – ROC curves showing anomaly detection performance for different time window sizes.

volatile time-series but also average out short-term anomalies which are then less likely detected. Furthermore, large time window sizes increase the detection time, i.e., the duration between an attack occurring and that attack being detected. This is due to the fact that an attack is always detected at the end of that time window, hence the average detection time is $\frac{tw}{2}$. Due to the fact that no knowledge about the attacks is available beforehand in practical applications, it is usually appropriate to fit the time window size according to periodic occurrences in the data.

Fig. 11 shows the ROC analysis for different settings of the time window size tw. For a reasonably low false positive rate, e.g., $FPR = 0.03$, smaller time window sizes < 60 min clearly outperform larger time window sizes. This is due to the mentioned issues that appear when the time window size is larger than the duration of the attacks. Furthermore, $tw = 7$ min causes a similarly poor performance due to the fact that it does not properly align with the present periodicity and thereby leads to highly sporadic cluster developments in all clusters that are affected by the corresponding log lines. The best possible choices for tw are therefore factors of the periodic interval, e.g., 15 min or 30 min.

In order to investigate the influence of the data set, a more complex log file was generated. In this data set, identical attacks were scheduled, i.e., the affected time intervals and the absolute number of anomalous log lines are the same as before. However, 5 additional users constantly produce log lines corresponding to normal behavior. This means that the attacks are more difficult to detect since the fraction of anomalous lines in every time window decreases. Fig. 12 shows the comparison between the ROC curves from the complex data set as solid lines and the previously computed ROC curves as dashed lines. As expected, performance on the more complex data set decreases due to the mentioned issues. Surprisingly,

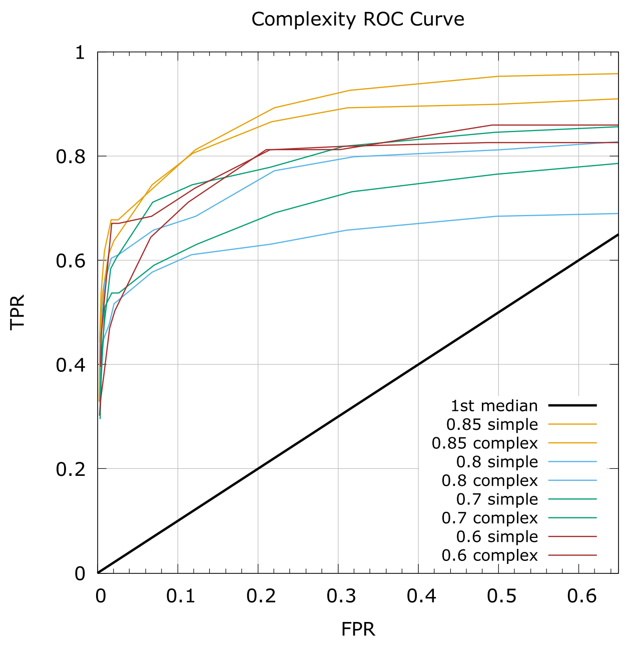


Fig. 12 – ROC curves showing the influence of data complexity on the anomaly detection performance.

$t = 0.7$ poses an exception to this pattern as the results improved on the complex data set. The reason for this is that the percentage of log lines contained in evolving clusters increased compared to the simple log file, meaning that more evolving clusters were available for anomaly detection. This effect was already mentioned in Section 7.3.1 and is linked to the fact that even though there are more users producing noise, the overall variability of the cluster sizes recorded at the end of each time window decreases and thus better

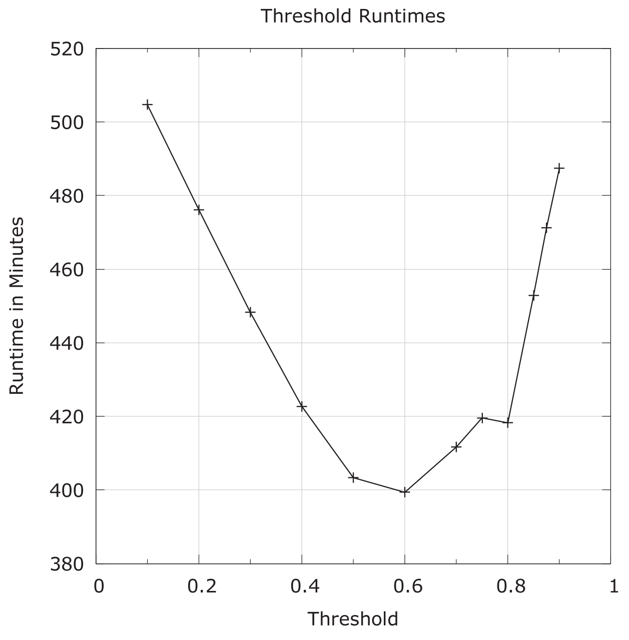


Fig. 13 – Total runtimes for different similarity thresholds.

predictions can be made. The decreased variability also means that it is less likely that the development of an evolving cluster is interrupted due to randomly occurring non-representative log lines within certain time windows.

7.5. *Runtime and scalability*

Since anomaly detection is often computationally intensive, it is important to consider the runtime and scalability in addition to *TPR* and *FPR*. Fig. 13 shows the recorded runtime for different similarity threshold values. Moderate values around $t = 0.6$ are clearly favorable over extreme values close to 0 or 1. Both clustering and time-series analysis are responsible for this effect. When a low t is used, smaller numbers of clusters are available and thus less time is spent approximating ARIMA models. However, the low t also causes that almost none of these clusters are eliminated as candidates and the computationally complex string distance metric has to be computed multiple times for every incoming log line. This effect disappears when t is high. However, due to the large amount of clusters, many ARIMA models have to be fitted, thereby increasing the runtime. For this reason, we were not able to carry out any experiments with $t > 0.9$ due to limitations in available processing power.

In order to ensure the ability to process data streams of arbitrary length, a linear scalability is required, i.e., the runtime must only linearly depend on the number of log lines. This characteristic was empirically verified by measuring the elapsed time after each set of 50,000 log lines. Fig. 14 shows these cumulated runtimes for different similarity thresholds. As it can be seen, the runtimes exhibit a linear behavior independent from the chosen threshold and despite the attacks being present in the data. We therefore argue that the proposed methodology allows processing any number of log lines as well as continuous log streams.

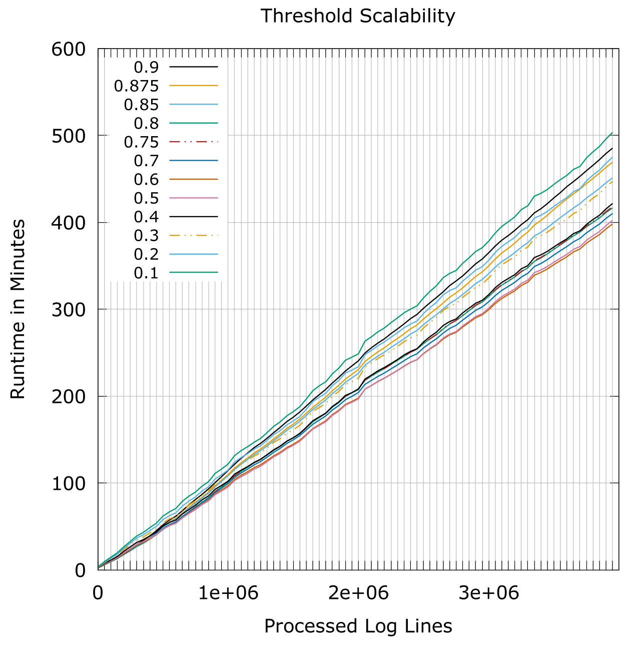


Fig. 14 – Plot showing the continuously measured runtime that is required for processing a certain amount of log lines. The runtime scales linearly for all considered similarity thresholds.

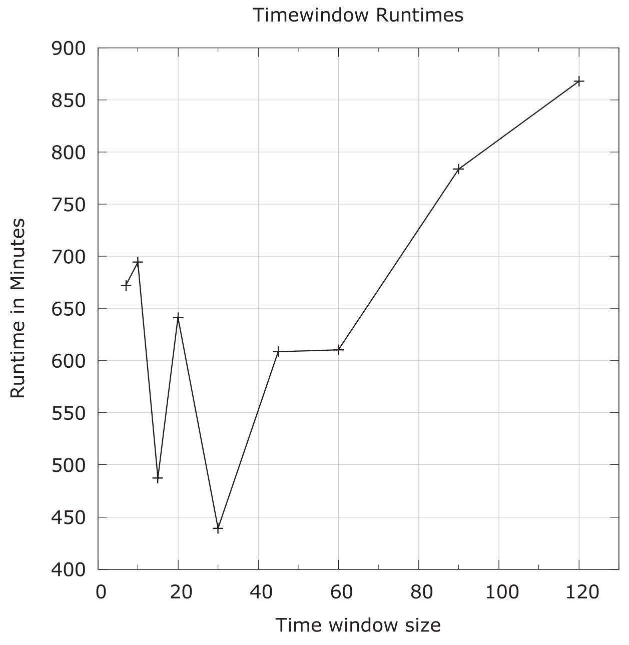


Fig. 15 – Total runtimes for different time window sizes.

The influence of the time window size on the runtime is investigated in a similar manner. Fig. 15 shows all runtimes that were measured for different values of tw. It can immediately be seen that the runtime increases for large time windows ($tw > 60$ min), while $tw = 30$ min and $tw = 15$ min show the lowest runtimes. Other than for t, the runtime is solely dependent on the time required for fitting the ARIMA model, while the time spent on clustering is largely independent from tw.

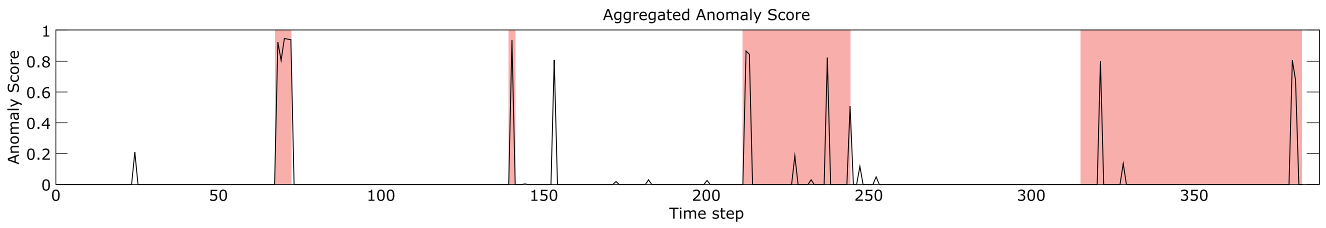


Fig. 16 – Anomaly score of every time step. The phases of occurring attacks are shaded in red.

The reason for this is that a fixed number of data points from the past are used for fitting the time-series model, independent from the selected time window size. For example, when 48 historic values are stored for fitting the time-series model, the last 48 h are used for computing the prediction intervals when $tw = 60$ min, but only 12 h are used when $tw = 15$ min. A smaller time window size therefore causes the algorithm to remove data from the past earlier and anomalies in the historic data thus have less influence on the predictions. It is also computationally faster to fit an ARIMA model on the anomaly-free data, thereby leading to shorter runtimes for smaller time window sizes.

An alternative implementation could only store data points that lie within a fixed duration in the past, e.g. the last 24 h. This has some obvious implications regarding the runtime. The smaller the time window size is selected, the more data points have to be considered when fitting the ARIMA model. Therefore, the runtime is likely to increase for short time windows and decrease for large time windows in this setting.

Finally, we investigated the scalability of the algorithm with respect to tw. For any setting, a linear behavior similar to Fig. 14 was achieved. The interpretation is thus identical to the previous scalability analysis. The plot is skipped for brevity.

7.6. *Aggregated detection*

Aggregating the detected outliers from all clusters gives an overview of the current state of the whole system. The anomaly score introduced in Eq. (16) is a measure for the deviation from the expected cluster sizes from all clusters that exist for at least 20 time steps. Fig. 16 shows the anomaly scores yielded in every time step using $t = 0.875$, $tw = 15$ min and $\alpha = 0.00001$. The value for α was chosen rather small in order to minimize the influence of false positives while emphasizing the large deviations occurring in specific clusters where the anomalies manifest themselves clearly. The intervals shaded red indicate the appearances and durations of the injected anomalies.

The plot confirms the previous observations regarding the successful detection of the first three anomalies. The anomaly relating to the long-term frequency increase once more shows very distinctly that the algorithm only detects changes of the system behavior, but immediately adjusts to shifts and trends. For that reason, the anomaly score within the shaded interval is mostly 0. Only when the system returns to the normal behavior, the anomaly score shoots up again. There further exist a few spikes outside of the shaded regions which are either false positives or artifacts from previous anomalies, e.g., the small spikes around time step 250.

In practice, the anomaly score may be used with an alarm threshold. In this scenario, a threshold around 0.4 yields reasonable results since all anomalies and only a single false positive are reported. The selection of this alarm threshold should be decided individually from observation of the system at hand. Alternatively, the anomaly score could again be treated as a time-series and analyzed with appropriate methods.

7.7. *Application on real log data*

While it is difficult to determine *TPR* and *FPR*, insights about the practical applicability can be derived from experiments with real log data. Therefore, the anomaly detection was applied on log data collected within the Austrian Institute of Technology (AIT). Both automatized processes that operate with different periodicities as well as erratic human behavior contribute to the captured logs. The logs were recorded over the course of 1 week without any interruptions. The following parameters were selected: $t = 0.8$, $tw = 30$ min, $\theta = 0.7$ and $\theta_{part} = 0.2$. With this setting, more than 90% of the total amount of log lines are successfully represented by evolving clusters that exist for at least 5 time steps. Fig. 17 shows an exemplary cluster size development that exhibits interesting characteristics. A time window size of 30 min means that 1 day is represented by 48 time steps. The patterns that are visible in the plot appear accordingly to this interval on the first, second, third and seventh day. As expected, these plateaus are correctly identified as anomalies. The ARIMA model was set up to recognize periodicities repeating within a maximum of 12 h and this pattern is therefore not learned.

Other plots exhibited artifacts that corresponded to the displayed cluster size but differed in shape and magnitude. Furthermore, some clusters captured the highly precise periodic behavior of scheduled programs or the noisy behavior of randomly interfering events.

Fig. 18 shows the aggregated anomaly score computed for all clusters that exist for at least a total of 100 time steps. Several of the spikes correspond to the artifacts visible in the cluster evolution plot. Furthermore, some anomalies from other clusters also yield a rather high anomaly score, e.g., at time step 150. It appears that the most anomalies occurred during the third day and only few anomalies occurred on the fourth, fifth and sixth day.

There was no known attack taking place during the time where the log was captured and the detected anomalies only correspond to harmless events such as updates. It is not sur-

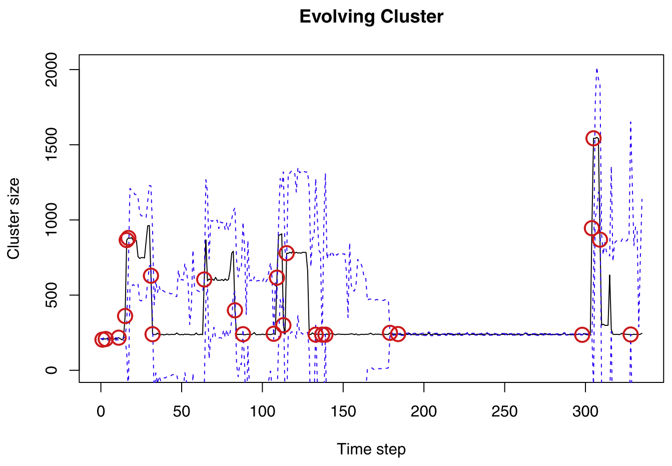


Fig. 17 – Development of a cluster size measured on real data.

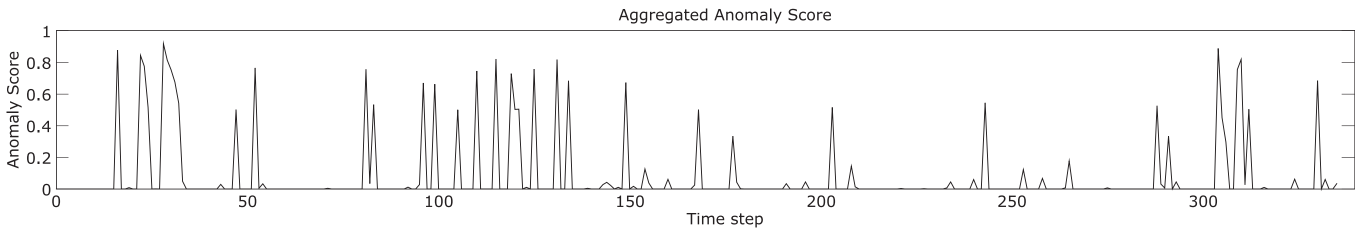


Fig. 18 – Anomaly score computed on real data.

prising that the algorithm detects such events as anomalies due to the fact that the attacks are assumed to manifest themselves in a similar way. There is thus no simple way for an algorithm to differentiate between an anomaly corresponding to an attack and an anomaly caused by regular events such as updates. This is an obvious drawback that affects unsupervised self-learning anomaly detection methods in general. Nevertheless, in critical systems the risk of occasional false alarms is accepted in order to ensure that attacks that are difficult to detect for other methods are not overseen. Furthermore, the knowledge of the human administrator about scheduled events that are possibly detected as anomalies should be sufficient to dismiss many of the false positives immediately.

8. Conclusion and future work

A methodology for dynamic anomaly detection in log files was introduced in this paper. At first, log lines are incrementally grouped by similarity in order to establish static cluster maps. Then, log lines are allocated to the existing cluster maps created in the preceding and succeeding time windows. Thereby, a connection between clusters of two neighboring cluster maps that previously did not share any common elements is established. This enables the computation of an overlap metric that measures the likelihood of a cluster from one cluster map transforming into another cluster from the succeeding cluster map and allows the detection of transitions such as splits or merges. Metrics are derived from the resulting cluster developments and approximated by ARIMA models. Deviations from expected behavior are detected using one-step ahead forecasts.

A semi-synthetically generated log file was used for evaluation. The anomaly detection showed promising performances when applied on the evolutions of individual clusters. Moreover, an aggregated anomaly score showed clear peaks when the attacks were injected in the system. Finally, evaluation on a real log file revealed anomalies corresponding to system behavior changes.

Several modifications that may have a positive influence on the overall performance are possible. The incremental clustering algorithm could be replaced by another machine learning technique that is able to group similar strings in an unsupervised manner. Despite higher computational requirements, the mentioned overlap metric that takes multiple time windows into account when determining the connections between clusters could result in more reliable evolutions. Furthermore, the time-series analysis could be carried out using other models than ARIMA. Detection on time-series could also be realized using filters rather than predictions. In addition, methods that employ change point analysis are promising solutions to detect anomalies that cause gradual and long-term changes (Killick et al., 2012). Especially the fourth injected anomaly that remained undetected in many clusters could successfully be identified by such techniques.

The evaluation focused on the size of the cluster as it directly represents the frequency of the corresponding log line types in the respective time windows and was thus appropriate to detect the injected attacks. However, many other useful metrics that were defined in Sections 5.2 and 5.3 were left out from practical analysis, even though they could be better fitted for special types of anomalies. Other than performing anomaly detection on time-series created on each of these features alone there is also the possibility to combine them and apply multivariate outlier detection.

Finally, the problem of rather high amounts of false positives that all anomaly detection techniques suffer from remains unsolved. It appears that every attempt to make the algorithm more robust against such influences at the same time restricts its ability of detecting certain types of anomalies. Trustworthy and up-to-date domain knowledge about the specific use case would be required in order to additionally support the self-learning methods in differentiating between normal behavior and an actual anomaly.

Acknowledgements

This work was partly funded by the ECSEL project SemI40 (692466) and the FFG project synERGY (855457) and carried out in course of a master thesis at the Vienna University of Technology.

REFERENCES

Amor NB, Benferhat S, Elouedi Z. Naive bayes vs decision trees in intrusion detection systems. In: Proceedings of the 2004 ACM symposium on applied computing, SAC '04. New York, NY, USA: ACM; 2004. p. 420–4.

Andreasson J, Geijer C. Log-based anomaly detection for system surveillance. Master's thesis; 2015.

Asur S, Parthasarathy S, Ucar D. An event-based framework for characterizing the evolutionary behavior of interaction graphs. ACM Trans Knowl Discov Data 2009;3(4) 16:1–16:36.

Bilgin CC, Yener B. Dynamic network evolution: models, clustering, anomaly detection. IEEE Netw. 2006.

Bródka P, Saganowski S, Kazienko P. Ged: the method for group evolution discovery in social networks. Soc Netw Anal Mining 2013;3(1):1–14.

Carmi A, Septier F, Godsill SJ. The Gaussian mixture MCMC particle algorithm for dynamic cluster tracking. In: Proceedings of the 2009 12th international conference on information fusion; 2009. p. 1179–86.

Chakrabarti D, Kumar R, Tomkins A. Evolutionary clustering. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; KDD '06; 2006. p. 554–60.

Chan J, Bailey J, Leckie C. Discovering correlated spatio-temporal changes in evolving graphs. Knowl Inf Syst 2008;16(1):53–96.

Chandola V, Banerjee A, Kumar V. Anomaly detection: a survey. ACM Comput Surv 2009;41(3) 15:1–15:58.

Chi Y, Song X, Zhou D, Hino K, Tseng BL. On evolutionary spectral clustering. ACM Trans Knowl Discov Data 2009;3(4) 17:1–17:30.

Chin SC, Ray A, Rajagopalan V. Symbolic time series analysis for anomaly detection: A comparative evaluation. Signal Process 2005;85(9):1859–68.

Cortez P, Rio M, Rocha M, Sousa P. Multi-scale internet traffic forecasting using neural networks and time series methods. Expert Syst. 2012;29(2):143–55.

Cryer J, Chan K. Time series analysis: with applications in R, Springer Texts in Statistics. Springer New York; 2008.

Esling P, Agon C. Time-series data mining. ACM Comput Surv 2012;45(1) 12:1–12:34.

Falkowski T, Bartelheimer J, Spiliopoulou M. Mining and visualizing the evolution of subgroups in social networks. In: Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence (WI 2006 main conference proceedings)(WI'06); 2006. p. 52–8.

Fiore U, Palmieri F, Castiglione A, De Santis A. Network anomaly detection with the restricted boltzmann machine. Neurocomput 2013;122:13–23.

Fu Q, Lou JG, Wang Y, Li J. Execution anomaly detection in distributed systems through unstructured log analysis. In: Proceedings of the 2009 ninth IEEE international conference on data mining. Washington, DC, USA: IEEE Computer Society; ICDM '09; 2009. p. 149–58.

Goh J, Adepu S, Tan M, Lee ZS. Anomaly detection in cyber physical systems using recurrent neural networks. In: Proceedings of the 2017 IEEE 18th international symposium on high assurance systems engineering (HASE); 2017. p. 140–5.

Goldberg MK, Hayvanovych M, Magdon-Ismail M. Measuring similarity between sets of overlapping clusters. In: Proceedings of the 2010 IEEE second international conference on social computing. Washington, DC, USA: IEEE Computer Society; SOCIALCOM '10; 2010. p. 303–8.

Greene D, Cunningham P. Multi-view clustering for mining heterogeneous social network data. Proceedings of the paper presented at the workshop on information retrieval over social networks, 31st European conference on information retrieval (ECIR'09). Toulouse, France, 2009.

Greene D, Doyle D, Cunningham P. Tracking the evolution of communities in dynamic social networks. In: Proceedings of the 2010 international conference on advances in social networks analysis and mining; 2010. p. 176–83.

Gupta M, Gao J, Aggarwal C, Han J. Outlier detection for temporal data. Morgan & Claypool Publishers; 2014.

He S, Zhu J, He P, Lyu MR. Experience report: System log analysis for anomaly detection. In: Proceedings of the 2016 IEEE 27th international symposium on software reliability engineering (ISSRE); 2016. p. 207–18.

Hill DJ, Minsker BS. Anomaly detection in streaming environmental sensor data: a data-driven modeling approach. Environ Model Softw 2010;25(9):1014–22.

Hyndman RJ. The difference between prediction intervals and confidence intervals. https://robjhyndman.com/hyndsight/intervals/; 2013. [Online; accessed 07-August-2017].

Jensen CS, Lin D, Ooi BC. Continuous clustering of moving objects. IEEE Trans Knowl Data Eng. 2007;19(9):1161–74.

Juvonen A, Sipola T, Hämäläinen T. Online anomaly detection using dimensionality reduction techniques for http log analysis. Comput Netw 2015;91:46–56.

Khalilian M, Mustapha N. Data Stream Clustering: Challenges and Issues. Proceedings of the International MultiConference of Engineers and Computer Scientists, 1; 2010. arXiv: 1006.5261

Killick R, Fearnhead P, Eckley IA. Optimal detection of changepoints with a linear computational cost. J Am Stat Assoc 2012;107(500):1590–8.

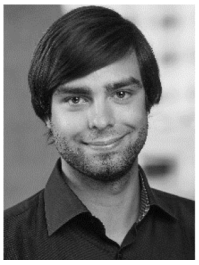
Kruegel C, Vigna G. Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM conference on computer and communications security; CCS '03. New York, NY, USA: ACM; 2003. p. 251–61.

Landauer M, Wurzenberger M, Skopik F, Settanni G, Filzmoser P. Time series analysis: unsupervised anomaly detection beyond outlier detection. Proceedings of the international conference on information security practice and experience, 2018.

Lee P, Lakshmanan LVS, Milios EE. Incremental cluster evolution tracking from highly dynamic network data. In: Proceedings of the 2014 IEEE 30th international conference on data engineering; 2014. p. 3–14.

Pincombe B. Anomaly detection in time series of graphs using arma processes. Asor Bull 2005;24(4):2.
Powers DMW. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. Journal of Machine Learning Technologies 2011;2:37–63.
Silva JA, Faria ER, Barros RC, Hruschka ER, Carvalho ACPLFd, Gama Ja. Data stream clustering: a survey. ACM Comput Surv 2013;46(1) 13:1–13:31.
Skopik F, Settanni G, Fiedler R, Friedberg I. Semi-synthetic data set generation for security software evaluation. In: Proceedings of the 2014 twelfth annual international conference on privacy, security and trust; 2014. p. 156–63.
Sperotto A, Sadre R, Pras A. Anomaly characterization in flow-based traffic time series. In: Proceedings of the 8th IEEE international workshop on IP operations and management; IPOM '08;. Berlin, Heidelberg: Springer-Verlag; 2008. p. 15–27.
Spiliopoulou M, Ntoutsi I, Theodoridis Y, Schult R. Monic: modeling and monitoring cluster transitions. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining; KDD '06. New York, NY, USA: ACM; 2006. p. 706–11.
Takaffoli M, Sangi F, Fagnan J, Zäıane OR. Community evolution mining in dynamic social networks. Procedia-Soc Behav Sci 2011;22:49–58.
Thottan M, Ji C. Anomaly detection in ip networks. IEEE Trans Signal Process 2003;51(8):2191–204.
Toyoda M, Kitsuregawa M. Extracting evolution of web communities from a series of web archives. In: Proceedings of the Fourteenth ACM conference on hypertext and hypermedia HYPERTEXT '03;. New York, NY, USA: ACM; 2003. p. 28–37.
Vaarandi R. A data clustering algorithm for mining patterns from event logs. In: Proceedings of the 3rd IEEE workshop on IP operations management (IPOM 2003) (IEEE Cat. No.03EX764); 2003. p. 119–26.
Wurzenberger M, Skopik F, Landauer M, Greitbauer P, Fiedler R, Kastner W. Incremental clustering for semi-supervised anomaly detection applied on log data. In: Proceedings of the 12th international conference on availability, reliability and security. ACM; 2017. p. 31.
Xu W, Huang L, Fox A, Patterson D, Jordan MI. Detecting large-scale system problems by mining console logs. In: Proceedings of the ACM SIGOPS 22Nd symposium on operating systems principles SOSP '09;. New York, NY, USA: ACM; 2009. p. 117–32.
Yassin W, Udzir NI, Muda Z, Sulaiman MN. K-means clustering and naive Bayes classification for intrusion detection. Proceedings of the 4th international conference on computing and informatics, 2013.
Zhou A, Cao F, Qian W, Jin C. Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst 2008;15(2):181–214.

Dipl.-Ing. Max Landauer finished his Bachelor's Degree in Business Informatics at the Vienna University of Technology in 2016. In 2017, he joined the Austrian Institute of Technology in 2017 where he carried out his Master Thesis. He started his PhD studies in 2018 and is currently employed as a Junior Scientist at AIT. His main research interests are anomaly detection and log data analysis.



DI Markus Wurzenberger finished his Bachelor's Degree in Mathematics in Science and Technology in 2013. In 2014 he joined AIT as a freelancer and finished his Master's Degree in Technical Mathematics in 2015. In the end of 2015 he joined AIT as Junior Scientist and is working on national and international projects in the context of anomaly detection. In 2016 he started his PhD studies in Computer Science.

Dr. Florian Skopik, CISSP, CISM, CCNP-S joined the Austrian Institute of Technology in 2011 and is the Thematic Coordinator of AIT's cyber security research program. He coordinates national and international (EU) research projects, as well as the overall research direction of the team. The main topics of his projects are focusing on smart grid security, the security of critical infrastructures and national cyber security. He published more than 100 scientific conference papers and journal articles and holds some 20 industryrecognized security certifications, Florian is member of various conference program committees and editorial boards, as well as standardization groups, such as ETSI TC Cyber and OASIS CTI. Florian is IEEE Senior Member, Member of the Association for Computing Machinery (ACM) and Member of the International Society of Automation (ISA).

Giuseppe Settanni, MSc, CISSP, joined AIT in 2013 and is currently working as a Scientist in the Centre for Digital Safety and Security. He holds a MSc degree in Telecommunication Engineering from Polytechnic University of Turin, Italy, as well as a Bachelor degree in Telecommunication Engineering from Polytechnic University of Bari, Italy. Before joining AIT, Giuseppe Settanni worked for 2 years at FTW (Telecommunication Research Center in Vienna) as a communication network researcher. His current research interests include security of critical infrastructures, information sharing, anomaly detection and cyber threat intelligence management for national defense. He has been involved in several national and EU-funded applied research projects concerning security in communication and information systems.

Dr. Peter Filzmoser is a Professor of Statistics at the Vienna University of Technology, Austria, and Head of the Research Group Computational Statistics. He received his Ph.D. and postdoctoral lecture qualification from the same university. He was a Visiting Professor at Toulouse, France and Belarus. Furthermore, he has authored more than 200 research articles and is a co-author of a book on analyzing environmental data (Wiley, 2008), on multivariate methods in chemometrics (CRC Press, 2009), and on compositional data analysis (Springer, 2018).