Unsupervised Anomaly Alerting for IoT-Gateway Monitoring using Adaptive Thresholds and Half-Space Trees

René Wetzig
Complex and Distributed IT-Systems
TU Berlin
Berlin, Germany
rene.wetzig@campus.tu-berlin.de

Anton Gulenko
Complex and Distributed IT-Systems
TU Berlin
Berlin, Germany
anton.gulenko@tu-berlin.de

Florian Schmidt
Complex and Distributed IT-Systems
TU Berlin
Berlin, Germany
florian.schmidt@tu-berlin.de

***Abstract*—Through increasingly powerful hardware, IoT-Gateways gain strong computational capabilities, enabling more sophisticated analytics, service deployments and virtualization possibilities. However, the difficulty of ensuring high reliability of IoT-Gateways and its deployed services increases along with the complexity of digital systems. Reliable operation of such gateways and devices therefore depends on automated methods for detecting abnormal behaviour as early as possible, in order to counteract the situation before the complete failure of a system component.**

We propose an unsupervised anomaly detection algorithm by extending the concept of Half-Space Trees through online adaptive thresholds for real-time computation. The anomaly detector consumes multidimensional time series data from an external monitoring component and is fixed in computational complexity to support real-time operation. In order to determine valuable hyperparameters, we propose an anomaly simulation system, providing options for pattern induced load simulations and different types of anomalies. Further, we evaluate the anomaly detection method on monitoring data of the open source software Project Clearwater, which is a light weight implementation of the IP-Multimedia Subsystem running on small sized VMs. Results show the applicability of the Half-Space Tree based anomaly detection with high detection rates (99.4%) and low number of false alarms ($<$ 3%).

***Index Terms*—anomaly detection; reliability; gateways**

I. Introduction

The rapid development of digitalization in the public and private sector opens opportunities for developing new smart and connected devices in many fields, including medical self-tracking, smart factories, home gardening, and many others. This development is mainly caused by the emerging connectivity driven by the 5G movement and through increasingly affordable usage of compute resources. Small sized barebone computers or smartphones are now capable of running multiple apps and services and even provide virtualization environments (e.g. Raspberry Pi running Kubernetes). There are many potential applications for using such devices as IoT-Gateways for forwarding sensor data or requesting computation-intensive services. However, powerful computation and virtualization capabilities involve new problems that arise from the deep technological stack and complex overall system architecture. These problems are exacerbated in the IoT domain, where high reliability is a prerequisite and often seen as a commodity.

Reliable operation of distributed small-sized devices and services includes a variety of tasks, including identification of problematic behaviour and security threats, finding the root cause in an anomaly situation, and the selection and execution of an appropriate counter measure. At the scale of todays IoT systems, these tasks are impossible to perform manually for human system administrators [1]. Therefore, automatic anomaly detection algorithms play an important role in the reliable operation of IoT infrastructures.

The domain of IoT-Gateways introduces special requirements and challenges when designing anomaly detection methods. First, in contrast to data centers and computation in the cloud, an IoT-Gateway has very limited resources. An anomaly detection algorithm on such a device must be able to analyse a data stream in a timely manner, while consuming a minimal amount of computational resources. Furthermore, the resources of an IoT-Gateway device are shared between productive code and the data analysis processes. Since the productive parts of the system have higher priority, the resources available for the anomaly detection are reduced even further.

The IoT landscape exhibits many types of anomalies, including intrusion detection, DDoS attacks, software bugs, hardware failures, etc. We define anomalies as degraded states of the system impacting the service quality, which the device shall deliver. Thus, we aim to detect anomalies before the service or devices fails, in order to allow countermeasures to resolve the underlying problem. In this work, we further limit the scope to resource anomalies like memory leaks, insufficient network bandwidth, or excessive CPU utilization.

The state of the art approach for detecting anomalous behaviour is based on expert-defined thresholds that trigger alerts, which are then evaluated by system administrators. The thresholds are applied on data from active or passive monitoring tools, which are commonly used in computing infrastructures. This approach has limited applicability to IoT-Gateways, where resource usage patterns vary and system load changes occur frequently. Manual thresholds are difficult to maintain and must be constantly adapted in order to guarantee a high detection rate and a low number of false alarms. This leads to stale alarm threshold definitions and a low quality of anomaly detection results.

In order to address these shortcomings, this paper introduces the following three key contributions:

1) We propose a novel, unsupervised anomaly detection approach for automated alerting of abnormal system behaviour. The goal of our approach is to detect abnormal behaviour before a component fails entirely. The approach is based on the machine learning algorithm Half-Space Trees (HS-Trees) [2] and is extended through combining HS-Trees with dynamic threshold functions [3], [4] to adapt to concept drifts and enabling online detection capabilities without a separate training phase with labelled data.
2) Further, we present an anomaly simulation environment for evaluating suitable hyperparameters for the proposed algorithm. The simulator produces a data stream by generating a varying system load and injecting simulated anomalies in varying frequencies. The implementation is based on the seasonal timeseries generator of the machine learning framework WEKA [5].
3) Lastly, we provide an evaluation on a testbed for simulating small sized IoT-Gateways. The resource usage of the simulated devices is monitored and analysed by the proposed anomaly detction algorithm. Anomalies are injected through a specialized anomaly injection framework, capable of creating memory leaks, resource fluctuations and further high load scenarios. The testbed runs an installation of Project Clearwater, an open source implementation of the IP Multimedia Subsystem, on several small sized virtual machines, which represent many IoT-Gateways.

The rest of the paper is structured as followed. Section II presents related approaches capturing the state of the art of further anomaly and state change detection methods. In Section III, we provide background about the HS-trees anomaly detection approach. Afterwards, we describe the extended Half-Space Trees anomaly detection approach in Section IV, while Section V shows an extensive anomaly simulation framework for hyperparameter optimisation. Section VI evaluates the unsupervised anomaly detection approach based on Clearwater service testbed. Finally, Section VII concludes this paper and gives an outlook for future work.

II. Related Work

Anomaly detection is a long standing and widely researched area with applications in diverse domains. The proposed approaches can be grouped into the two different categories, supervised and unsupervised methods.

Supervised approaches use labelled system information to train the machine learning models. Basically, the anomaly detection algorithms use either classification or regression models trained by data containing the information whether the data point is an anomaly or not.

Sauvanaud et al. [6] examine in their work a supervised ensembler approach for anomaly detection. They combine several different supervised classification algorithms, performing anomaly detection for individual services by using weighted voting mechanism. Thus, the collection of algorithms predicts whether a component is normal or not. The evaluation is based on a Virtual Network Function scenario similar to our setup as presented in this work. Also, Liu et al. [7] propose a supervised anomaly detection strategy. As bases, they use the concept of self organizing maps (SOM). The SOM is configured to represent the whole infrastructure in order to predict the overall performance of the system. Based on this model, they show that anomalies within virtual machines can be detected in related regions of the infrastructure. In contrast to our work, we do not aggregate data from several hosts in a single model, but create models per individual host in order to capture its own behaviour.

As supervised approaches consumes labelled training data, those are mostly insufficient to be provided by expert-labelling as the manual labelling is too time consuming. Automatic creation of data sets assume, that one can collect training data from a running system. This would potentially harm the productive system and is therefore undesirable. For practical usage, unsupervised methods are therefore investigated, having the positive properties of performing the same task, but not using labelled input data. Thus, our work aims also an unsupervised detection technique.

Dean et al. [8] are also using SOMs, but propose an unsupervised detection model. They aim to predict failures within a cloud infrastructure, but do not consider degraded state anomalies as use case. In contrast to these deeplearning models, Cotroneo et al. [9] use simple statistical correlation analysis between system components in order to automatically detect anomalies. Changes within these correlation are recommended as anomalies. Again, the focus of Cotroneo et al. is to combine data from different hosts. This approach suffers of scalability issues, which dynamic infrastructure introduces. Thus, we aim to create unsupervised detection models for individual hosts, which can be applied directly on the monitored entity. Chandola et al. [10] aggregate further concepts for anomaly detection techniques used in this field in a survey.

III. Half-Space Trees Background

Streaming Half-Space Trees (we will refer to it by *HST*) is an online anomaly detection algorithm proposed by Tan et al. [2], capable of unsupervised learning on multidimensional data-streams with continuous numerical attributes. It requires the data-stream to have constant dimensionality, prior knowledge of the number of dimensions as well as any dimension's upper and lower bounds. We will consider data points within a stream as arrays of fixed size and refer to the space spanned by the stream's known bounds as the domain.

The basic idea of the algorithm is to split the domain into a series of nested sub-spaces, or regions, along with a complete binary tree structure, called Half-Space Trees [2]. Each node of one of these binary trees has a part of the original domain associated to them, which we will call the node's work space. If a node has children, its work space is split into disjunct, equal-volume halves along a hyperplane bisecting a dimension chosen at random and each half is assigned to one of them. The work space of the root contains the entire domain. Thus, any data point in the domain can traverse a unique path from the root of a HST to one of its leaves along the work spaces it is contained in.

The data-stream is then also partitioned into short *windows* of a set length of $w \in \mathbb{N}$ data points. At any given time, we only consider two consecutive windows, the current one, called latest window, and the preceding one, referred to as the reference window. Once w data points have been recorded in a latest window, it is considered full, and becomes the reference window. With the next arriving data point, a new latest window begins. A data stream's structure is recorded as a *mass profile* [11], where each node has two counters r and l: r for the *reference mass profile* and l for the *latest mass profile*. A node's latest mass profile counter l is incremented by one when a newly arrived data point traverses it on its path to a leaf. Once the latest window is full, the reference mass profile counters of every node are overwritten with the value of their respective latest mass profile counters, which are then reset to 0. See figure 1 for a visual representation of a streaming HST and a recorded latest mass profile.

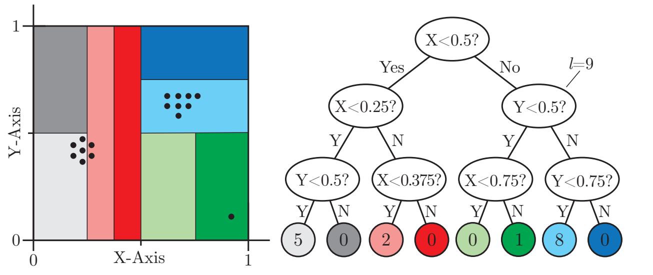

Fig. 1. Right: An HS-Tree of depth 3. Inner nodes contain their splitting dimension and value, leaves contain counters for the latest window mass profile. The root's right child also has the value of its latest mass profile counter marked. Left: Both dimensions of the two-dimensional domain is bounded by $[0,1]$ and is partitioned by the HS-Tree. Points represent data points. (Figure based on figure 1 in [2])

To aid the algorithm's robustness, an ensemble of $n \in \mathbb{N}_{>0}$ Half-Space Trees is created. To facilitate further diversity, the workspaces assigned to each tree's root are not the domain but a random perturbation of it which must contain it.

Anomaly prediction is performed by assigning an *anomaly score* to each data point upon its arrival. A score is calculated independently for each tree in the ensemble, the final anomaly score is the sum of all of these scores. We will refer to the score calculated for a data point x on a single tree T by $S(x,T)$, while denoting the final anomaly score over all trees with $S(x)$. $S(x,T)$ is calculated by letting the data point x traverse the tree T until a node is reached that is either a leaf, or has a reference mass profile equal to or less than a user-determined variable s as size limit. Tan et al. suggest a value of $\frac{1}{w}$, which we will also use for our evaluations. We will refer to that final Node by $Node^*$, we refer to the depth of a node in the tree by $Node.depth$ (note: roots have depth 0) and to its reference mass by $Node.r$, and can now calculate

$$S(x,T) := Node^*.r \cdot 2^{Node^*.depth}.$$

Due to the limit of s, and the different weights applied to a score according to the depth to which a new data point traversed, smaller variations in a data stream's distribution will not lead to an immediate drop of new data point's anomaly scores to 0. While this allows the algorithm to be applied to data streams with concept drift, it also means that anomalous data points may not result in anomaly scores of 0.

The algorithm itself needs few parameters. The authors describe its performance on a given data stream as somewhat independent of the algorithm's parametrisation.

Tan et al. [2] propose recording all data points along with their corresponding anomaly scores and sorting them by their anomaly scores in descending order. The anomalous data points may then be found at the bottom of the list. Accordingly, a prediction for any given data point can only be made once the data stream has ended, as the data just can be ranked by this approach. Also, the percentage of anomalies in the stream must be known in advance in order to determine the number of data points at the bottom to predict as anomalous.

This approach also relies on the assumption that anomalies are rare and fleeting, which we also assume. Continued and homogeneous anomalies will eventually be recorded in the reference mass profile and be predicted as normal.

IV. Unsupervised Anomaly Detection using Adaptive Thresholds

We propose a method that allows us to predict whether any given data point in the stream is anomalous immediately after its anomaly score has been calculated by introducing adaptive thresholds generated on the basis of the anomaly scores calculated for preceding points in the data stream.

We assume that thresholds viable for diverse data sets must be adaptive to allow for changes in the distribution of the anomaly scores when concept drift in the data stream sets in or ceases. We also assume that the anomaly scores of HST can generate for any given data stream are normally distributed, also assumed by [4], [12], [13]. Thus, the main idea relies on constructing a dynamic, point-wise threshold based on normal behaving anomaly scores.

Based on these assumptions, we have extended HST by introducing a threshold that is updated iteratively with each new anomaly score, generated using Tony Finch's Exponentially Weighted Moving Standard Deviation (EWSD) [14]. We compare a new anomaly score with the latest threshold. If the new anomaly score is lower than the threshold, we predict the data point associated with the anomaly score as anomalous. This allows us to make a prediction for a newly arrived data point immediately after its arrival. We then update the threshold using that latest anomaly score t.

The EWSD is calculated on the basis of the Exponentially Weighted Moving Average (EMA) proposed by J. Stuart Hunter [15]. The EMA generates a weighted average by applying weights, which decrease exponentially for older values and allows continuous introduction of new values without impact on performance. The EWSD is an equally weighted standard deviation that's efficiently updated along with the EMA. To facilitate the calculation of the EWSD, we will also introduce the Exponentially Weighted Moving Variance, which we will call *EMVar*.

The user can choose a variable $\alpha \in (0,1)$ that determines the weighting. We will refer to the current EMA after the i-th data point of the stream x_i has been inserted by EMA_i, the corresponding EMVar by $EMVar_i$ and the EWSD by $\widetilde{\sigma}_i$. Let $EMA_1 := S(x_1)$, $EMVar_1 := 0$ and $\widetilde{\sigma}_1 := 0$. Every further value is calculated by

$$\begin{aligned}\delta_i &:= S(x_i) - EMA_{i-1}\,,\\ EMA_i &:= EMA_{i-1} + \alpha \cdot \delta_i\\ &= \alpha \cdot S(x_i) + (1-\alpha) \cdot EMA_{i-1}\,,\\ EMVar_i &:= (1-\alpha) \cdot (EMVar_{i-1} + \alpha \cdot \delta_i^2)\,,\\ \widetilde{\sigma}_i &:= \sqrt{EMVar_i}\,.\end{aligned}$$

The current threshold t is then calculated using a user-determined constant $\eta \in \mathbb{R}_+$, and constantly overwritten after each new anomaly score has been inserted, according to the following formula:

$$t := EMA_i - \eta \cdot \widetilde{\sigma}_i\ .$$

We have decided against only using anomaly scores predicted as normal to update EMA and EWSD to allow for fast recovery in cases where our threshold might mistakenly interpret concept drift as anomalies.

V. Parameter Optimisation

In order to evaluate suitable hyperparameters for our anomaly detection approach, we propose a method for data stream simulation that enables researchers to test their anomaly detection algorithms on a locally generated simulated data stream. Our method allows for a wide variety of settings that let users evaluate their approaches' robustness by testing them against simulated data streams of varied complexity.

Our data stream simulator allows us to simulate a multidimensional data stream with concept drift. In our implementation, we concentrated on certain aspects of our proposal that were best suited for a data stream based on assumptions about the expected properties of an IoT-Gateway load usage, in order to find parameters for our adaptive thresholds that would yield a robust anomaly detection algorithm capable of working on a wide range of data streams.

A. Anomaly Event Simulator

We propose a data stream simulator that incorporates both, regular system patterns and anomalies. It outputs one data point at a time upon request. The features of each dimension of the data stream drift at a constant rate, which may differ between dimensions. We assume that anomaly detection algorithms that, like HST, require the require the domain of the data stream to be bounded in every dimension, are independent of the actual bounds, since we can normalise each dimension's domain to the same interval $[x,y]$ with $x,y \in \mathbb{R}$ using a bijective mapping. Due to this and the mechanisms by which HST splits the domain into regions and records a data stream's structure, our simulated data stream's domain can have homogeneous upper and lower bounds in each dimension without compromising our results' applicability to real-world data. Table I shows and describes the different parameters for the anomaly datastream generator.

TABLE I
Parameters that determine the structure of a simulated data stream.

nrOfDims	Number of dimensions in the data stream.
min & *max*	Upper and lower bound for all dimensions.
minNormal & *maxNormal*	Upper and lower bound of the "tunnel" within which normal data points fluctuate.
minStepSize & *maxStepSize*	Minimal and maximal rate of fluctuation in a dimension per data point, ex. 0.1 for $0.1 \cdot \lvert max - min \rvert$.
anomalyPercentage	Anomaly rate in the data stream.
anomalyLength	Length of an anomaly (number of data points it lasts for).
firstAnomalyDim	First dimension of the data stream to be affected by anomalies.
anomalyDims	Number of dimensions affected by an anomaly. Anomaly dimensions occur in a block starting with *firstAnomalyDim*.
randomise	Boolean that determines whether fluctuation rate will be chosen at random from $[minStepSize, maxStepSize]$, or decrease linearly between dimensions 1 and *nrOfDims* from *maxStepSize* to *minStepSize*.
cleanPoints	Number of clean data points at the beginning of the stream. May be needed for unsupervised anomaly detection algorithms.
sinus	Boolean that determines whether values of each dimension follow a sinus curve or an angular trajectory.

Normal system behaviour is generated by mainly the decision about *sinus* and the *minStepSize* and *maxStepSize* determining the rate of changes. For example, the angular trajectory pattern describes normal points as: The initial data point is defined on all dimensions as $min + \lvert max - min \rvert \cdot minNormal$. Succeeding data points change randomly by uniform distribution selected interval inclusively between *minStepSize* and *maxStepSize* until reaching $min + \lvert max - min \rvert \cdot maxNormal$. The direction is then reverted, such that it aims again to the initial starting point. This behaviour is shown in Figure 2.

Once *cleanPoints* have been generated, a random number generator is used to inject anomalies in the dimensions with index *firstAnomalyDim* to this index plus *anomalyDims*. Through the probability of *anomalyPercentage*, the anomaly generator decides point-wise when an anomaly will included

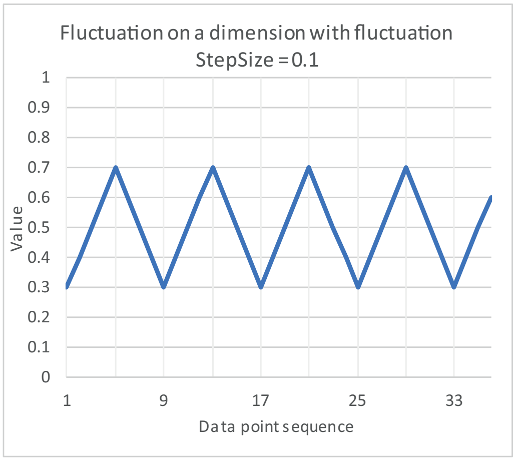

Fig. 2. Example for fluctuation on a single dimension of a data stream with $StepSize = 0.1$, $min = 0$, $max = 1$, $minNormal = 0.3$, $maxNormal = 0.7$.

within the data stream. Thus, when an anomaly happens, the anomaly gets the value *min* or *max* dependent on which value is in the current stream further away. The value *anomalyLength* describes for how many data points the anomaly appears in a block. Figure 3 shows such a point anomaly on an angular trajectory pattern.

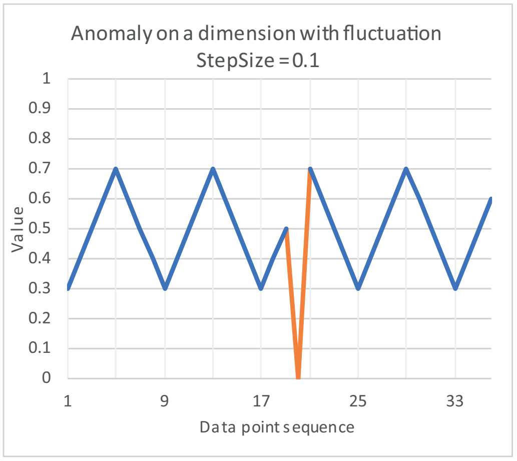

Fig. 3. Example of an anomaly on the dimension from figure 2

B. Parameter Evaluation

For hyperparmeter evaluation, we generated a data stream using the setup parameters shown in Table II. Furthermore, we focused on the different hyperparameters in order to identify key findings for hyperparameter setup for our dynamic threshold extension, as Tan et al. [2] provided in their paper base setups, which we also use. Therefore, the window size is set to $w = 250$, size limit to $l = 0.1 \cdot w$, number of trees $n = 25$ and the maximum depth of a tree is set to $d = 15$.

C. Setup for HST+EWSD Parameter Search

Given this setup, Figure 4 illustrates a scatter plot, representing the anomaly scores over time. Blue dots represent normal behaviour, while the orange dots represent anomalies. As expected, low scores indicate higher change to see abnormal data points. But a clear boarder seems not to be easily be defined, which can be also seen in Figure 5, showing the histogram of the sample window scores.

TABLE II
PARAMETERS FOR THE DATA STREAM SIMULATOR WE'VE CHOSEN FOR FINDING GOOD PARAMETERS FOR OUR HST+EWSD ALGORITHM.

nrOfDims	30
min & *max*	0 & 1
minStepSize & *maxStepSize*	0,001 & 0,1
minNormal & *maxNormal*	0,3 & 0,7
anomalyPercentage	2 %
anomalyDims	2
anomalyLength	1
firstAnomalyDim	15
randomise	FALSE

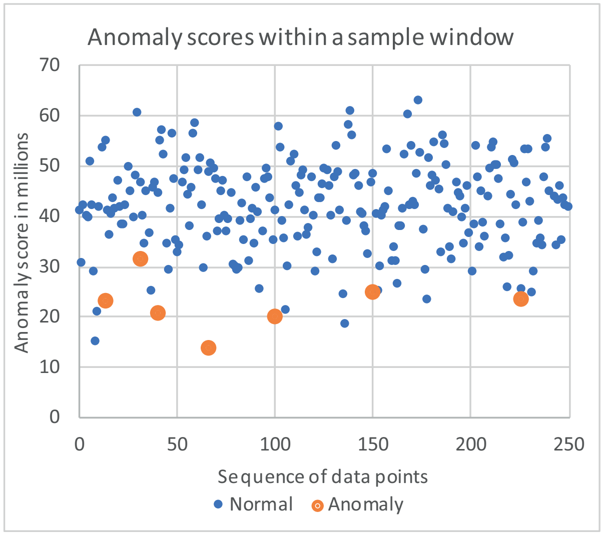

Fig. 4. A visualisation of anomaly scores within a sample window. Smaller blue dots represent anomaly scores of normal data points, larger orange dots represent those of anomalies we have injected.

Thus, for further threshold evaluation we examine grid search of the hyperparameters α and η. Here, we extensively evaluated large setups. For this paper, we just show the key frames from the grid search indicating highest value of information to chose a suitable parameter. Thus, we show next the results of a grid search in the intervals $\alpha = [0.06, 0.18]$ with step size 0.04 and $\eta = [0.6, 1.8]$ with step size 0.4.

D. Parameter Results

For evaluation purposes, we selected the true positive rate (relative number of anomalies, which are also detected as anomalies) and true negative rate (relative number of correctly classified normal data points out of all normal data points). Figure 6 shows the results for both parameters. The image indicates by increasing the values for both parameters, the higher the better normal data points are detected, while anomalies are not recognised as good anymore. At the point of intersection

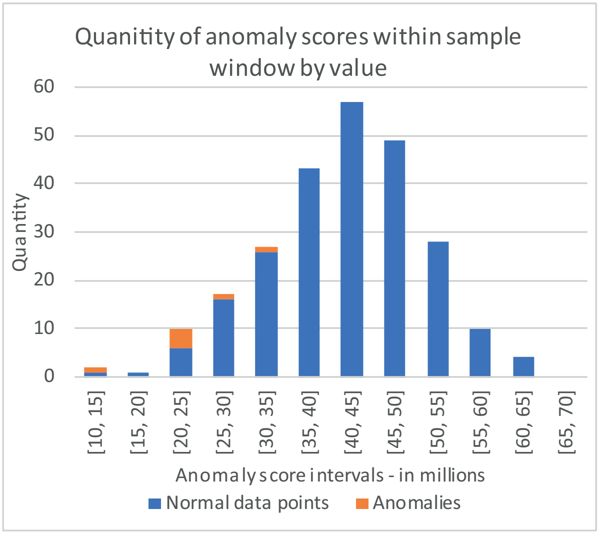

Fig. 5. Empirical distribution of data points in figure 4.

($\alpha = 0.06$, $\eta = 1.0$), most promising results are shown with both rates above 80%.

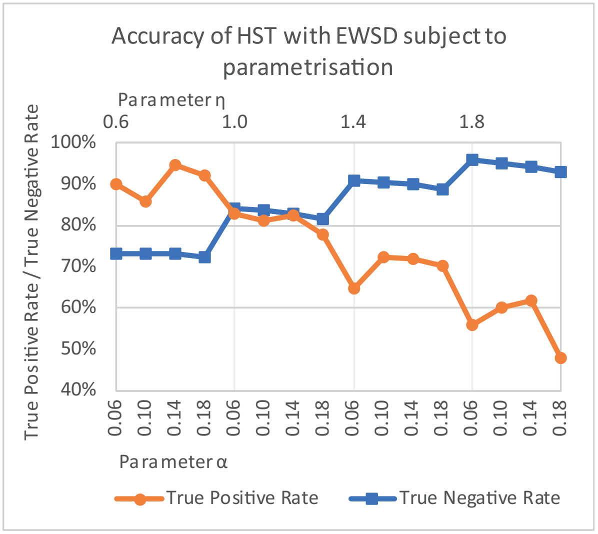

Fig. 6. Accuracy of the HST+EWSD algorithm subject to differing parametrisation of the adaptive thresholding algorithm.

Based on these hyperparameters, we show next an evaluation based on a service testbed, running light weighted software on a small sized virtual machine, representing a small sized IoT-Gateway.

VI. IoT-Gateway Evaluation

For evaluating the applicability of our solution to IoT-Gateway devices, we present next a testbed simulating typical service components running on an Gateway device utilising available open source solutions. The service components are described next in detail, followed by the infrastructure description.

A. Service Setup

We deployed the open source implementation of an IP multimedia subsystem (IMS), called Clearwater[1]. Clearwater is considered as one of the first examples for virtual network functions, which makes it highly interesting to telco-providers, considering services in the access network or even on gateway devices. IMS is an emerging architecture for IP-based telecommunication services, such as voice-, video calls or messaging. The core implementation of Clearwater allows users to register, and handles the initiation process to connect calls. This initiation process is mainly retrieving information about the user authentication and querying users connection details, which are important for initiating the call. The call itself is not handled anymore by Clearwater. There exists the services named Bono, Sprout, Homestead and Ellis, which are considered within the experiment.

Bono functions as edge proxy, handling clients connections. Those can register via the SIP protocol in order to initiate calls. The requests are then routed to the Sprout service.

Sprout manages the different communications to the other internal services, e.g. requesting for authentication.

Homestead contains the client profile information, which are needed to authenticate clients.

Ellis obtains the information for the management unit, such it functions as an account management system.

In order to provide realistic results, we simulate the usage of the IMS by changing every minute the number of client registrations and call initiations randomly between 20 and 40 users. Furthermore, we deployed a replicated version, which is load balanced. Thus, the key services Bono and Sprout are deployed three times each, while single deployments are used for Homestead and Ellis.

B. Infrastructure Setup

We use as cloud infrastructure a balanced and replicated Openstack[2] installation. Through Openstack, it is possible to create virtual machines, in which we deploy the above described services. For each service installation, we created its own virtual machine. The virtual machines run Ubuntu 14.04 and use 2 vCPU cores, 2GB memory and 20GB disk.

For monitoring, we collect inside each virtual machine the resource usage data parsing the proc filesystem, provided by the operating system. The data collection interval is 500ms.

C. Anomaly Injection

In order to evaluate the proposed approaches, we simulate a larger set of resource anomalous behaviours in the system. Thus, we developed an injection agent handling several different anomalies. The agent is placed inside each virtual machine and can execute the following anomalies:

- *Disk* pollution, temporary disk pollution and HDD stress: Both anomalies are writing data into a log-file. For the temporary case, the files are deleted, while in the other

[1]http://www.projectclearwater.org/
[2]https://www.openstack.org/

case the file is still available. HDD stress describes an excessive hard disk usage by writing a file.

- *CPU* stress, leak and fluctuation: CPU stress immediately consumes excessively much CPU. CPU leakage describes an anomaly increasing in its intensity over time by consuming more and more CPU. The fluctuation of CPU constantly increases and decreases the CPU usage in order to provide a fluctuation behaviour.
- *Memory* stress, leak and fluctuation: Like the CPU, memory anomalies are also modelled with the same behaviours: immediately consuming high amounts of memory, growing consumption over time and fluctuating allocation of memory.
- *Fork* flooding leak and fluctuation: A process starts to create child processes over time. For the leakage variant, those child processes create more processes, while in the fluctuation variant children are also removed partly.
- *Large file* download: A process is started downloading a large file, resulting in high network traffic.
- *File pointer* wasting: By requesting file pointers without releasing them, creates a leakage of the pointer IDs over time.

Normal behaviour (NR): After introducing all the different anomalies, there will be always phases between those, where the whole system runs without any anomaly, which we consider as normal behaviour.

Through using several simulated anomalies with different behaviours, we like to show the robustness of our anomaly detection approach. Notably, the intensity increase of each anomaly can be configured and is randomly selected for the current evaluation.

D. Analysis Pipeline Configuration

As described above, we collect metrics from the proc filesystem, which consists of a 29 collected values. Before performing the anomaly detection on the data, we do the following as preprocessing step min-max scaling in order to normalise the data in a fixed range $[0,1]$. Min-max scaling is performed for each metric, resulting in a scaled metric $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$ for an incoming data point x, where x_{min} and x_{max} represent the minimum and maximum values for the individual metrics. These can be inferred over time or set by an expert.

Based on the evaluation setup, we collected data of all services for 72h. The first 20 minutes of data consists of normal data grace time. Afterwards, the anomalies were injected in a round-robin manner to the individual service hosts for 3 minutes. Between the execution of anomalies, a cool down of 1 minute is performed. For evaluation purposes, we used datasets for each individual host, containing anomalies running on that specific host and time frames where the system runs in normal mode.

E. Evaluation Results

For more meaningful evaluation results, we decided to use a block based evaluation as we aim mostly to detect anomalies

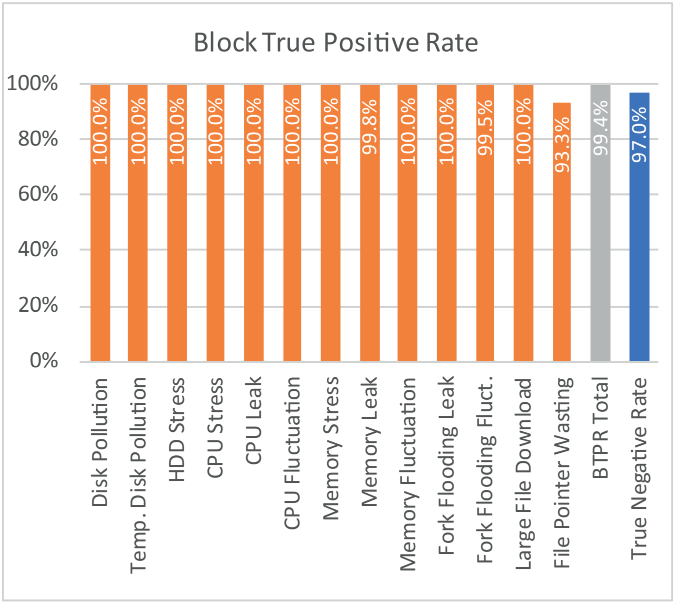

Fig. 7. Detection accuracy of the HST+EWSD algorithm on an the IoT-Gateway data stream.

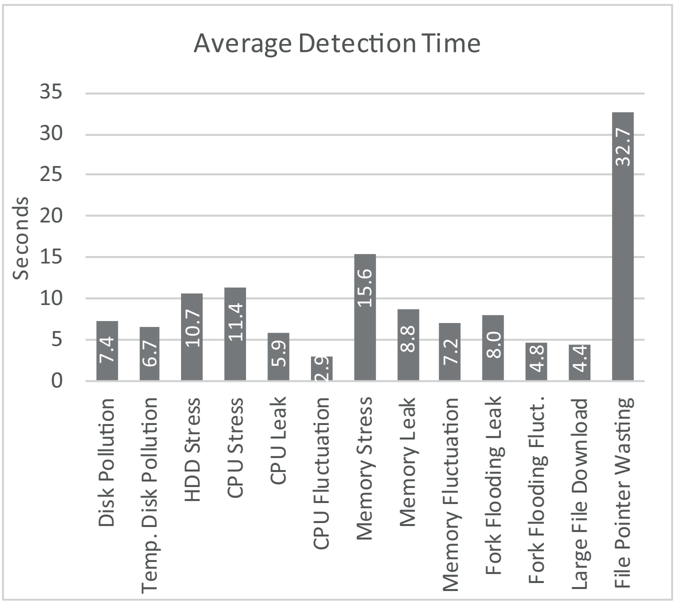

Fig. 8. Average anomaly detection time of the HST+EWSD algorithm on an IoT-Gateway data stream.

We measure the percentage of detected anomaly events. An anomaly event is a block of a single typed anomaly, which appears for a certain amount of time. In our setup, such blocks are of size 3min, thus contain 360 data points due to the monitoring interval. Thus, for each anomaly type we have 56-59 blocks of anomalies in total. Furthermore, we consider the detection time, when the anomaly was detection compared to the initial start when the injection of the anomaly was initiated. Additionally, we consider the point-wise true negative rate, which indicates how accurate the normal data points where detected and therefore we can induce the false alarm rate from it.

Figure 7 shows the detection results and the true negative rate. The results show (in orange), that almost all abnormal events were detected ($> 99\%$), except the anomaly file pointer wasting (93.3%). Through detailed investigation, it seems to be difficult for even administration experts to see this abnormal behaviour within the data. We therefore assume, that for this type of anomaly, more precised metrics need to be included or further feature engineering to provide data, which can be used to detect such type of anomaly. Over all (in grey) the detection rate of all anomaly events (BTPR Total), the results show strong results with 99.4%.

More details about the time until the anomaly was detected after starting those, Figure 8 gives insights about which anomalies where detected most immediate. Likewise to the anomaly detection rate, file pointer wasting shows least accurate performance. With 32.7sec detection time (not including those runs, where it was not detected at all), the other anomalies can be found much earlier.

The true negative rate (Fig. 7 blue) shows also high qualitative results with 97%, but still provides room for improvements as 3% of data points are false alarms. E.g. telco-providers require a reliability of 99.999% [16] in order to provide production ready solutions to meet their aimed SLAs.

VII. Conclusion

This work presented an unsupervised anomaly event detection approach, which extended the Half-Space Tree algorithm from Tan et al. [2] by dynamic threshold definition, enabling point-wise detection. We showed how valuable hyperparameters can be found by introducing a anomaly data stream simulation framework. Furthermore, the anomaly detection algorithm was evaluated using a testbed simulating IoT-Gateway devices through small sized virtual machines running an IMS service use case. Through injecting different kinds of resource anomalies, showed that the approach achieves high detection rates ($> 99\%$) for all anomalies with a small number of false alarms ($< 3\%$).

In future, we like to evaluate this approach on further services usecases and extend the evaluation on different sized virtual machines in order to provide more insights about computation bottlenecks with respect to resource usages. Furthermore, we like to investigate in more detail different device usage patterns to simulate more accurate system load. Furthermore, we envision to provide extensive study about different anomaly detection algorithms to provide the possibility for future zero touch administration.

References

[1] M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing qos architecture: Analysis of cloud systems and cloud services," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 1, pp. 6–18, 2017.

[2] S. C. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[3] F. Schmidt, A. Gulenko, M. Wallschläger, A. Acker, V. Hennig, F. Liu, and O. Kao, "Iftm-unsupervised anomaly detection for virtualized network function services," in *2018 IEEE International Conference on Web Services (ICWS).* IEEE, 2018, pp. 187–194.

[4] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 387–395.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[6] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Anomaly detection and root cause localization in virtual network functions," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on.* IEEE, 2016, pp. 196–206.

[7] J. Liu, S. Chen, Z. Zhou, and T. Wu, "An anomaly detection algorithm of cloud platform based on self-organizing maps," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[8] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012, pp. 191–200.

[9] D. Cotroneo, R. Natella, and S. Rosiello, "A fault correlation approach to detect performance anomalies in virtual network function chains," in *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on.* IEEE, 2017, pp. 90–100.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[11] K. M. Ting, G.-T. Zhou, F. T. Liu, and J. S. C. Tan, "Mass estimation and its applications," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 989–998.

[12] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[13] D. Oh and I. Yun, "Residual error based anomaly detection using auto-encoder in smd machine sound," *Sensors*, vol. 18, no. 5, p. 1308, 2018.

[14] T. Finch, "Incremental calculation of weighted mean and variance," *University of Cambridge*, vol. 4, no. 11-5, pp. 41–42, 2009.

[15] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[16] E. Bauer, X. Zhang, and D. A. Kimber, *Practical system reliability*. John Wiley & Sons, 2009.