

Anomaly Detection Within Mission-Critical Call Processing

Sean Doris¹ Iosif Salem² Stefan Schmid^{2*}

¹Motorola Solutions, Inc., 2600 Glostrup, Denmark
sean.doris@motorolasolutions.com

²TU Berlin, 10587 Berlin, Germany
iosif.salem@inet.tu-berlin.de, stefan.schmid@tu-berlin.de

Abstract

With increasingly larger and more complex telecommunication networks, there is a need for improved monitoring and reliability. Requirements increase further when working with mission-critical systems requiring stable operations to meet precise design and client requirements while maintaining high availability.

This paper proposes a novel methodology for developing a machine learning model that can assist in maintaining availability (through anomaly detection) for client-server communications in mission-critical systems. To that end, we validate our methodology for training models based on data classified according to client performance. The proposed methodology evaluates the use of machine learning to perform anomaly detection of a single virtualized server loaded with simulated network traffic (using SIPp) with media calls. The collected data for the models are classified based on the round trip time performance experienced on the client side to determine if the trained models can detect anomalous client side performance only using key performance indicators available on the server.

We compared the performance of seven different machine learning models by testing different trained and untrained test stressor scenarios. In the comparison, five models achieved an F1-score above 0.99 for the trained test scenarios. Random Forest was the only model able to attain an F1-score above 0.9 for all untrained test scenarios with the lowest being 0.980. The results suggest that it is possible to generate accurate anomaly detection to evaluate degraded client-side performance.

Keywords: anomaly detection, call processing system, mission-critical systems, virtualized environments, machine learning, random forest

1 Introduction

Anomaly detection has become a key focus for many industries as a method of producing accurate monitoring and analysis. Their applications range from assisting medical diagnosing and establishing a prognosis of cancer [28] to monitoring the system health of servers and routers in various telecommunication applications [13]. While the applications differ, many anomaly detection applications are interested in being used in mission-critical applications where safety and reliability are essential. The use of machine learning (ML) in these scenarios allows for improved data analysis by detecting complex patterns that could have otherwise gone unnoticed and could be used as predictors for future failures.

Mission-critical telecommunication systems have become increasingly advanced to handle the growing demand for higher data throughput and scalability. New features and bug fixes are regularly being deployed through releases of updates putting systems in a state of flux. Scalability is often implemented using several multi-card chassis, each having numerous virtual machines or containers to handle call volume [19]. All the while, the systems must be thoroughly tested to ensure that telecommunication systems have a high degree of reliability to ensure regular availability. Various potential failures can occur on these systems, such as reduced link quality, resource contention, orphan processes, hardware failures, and thermal limiting. Even minor hard-to-detect failures could cause performance degradation beyond safe operating conditions. Furthermore, when one virtualized

*Supported by German Research Foundation (DFG) project ReNO (SPP 2378), 2023-2027.

instance fails, it can affect the performance of other connected virtualized instances by increasing response times and delays [18]. This degradation can be difficult to diagnose in a large, complex, modularized network. Modularizing systems using virtual machines and docker containers allows for improved maintainability and load balancing [4].

Mission-critical applications' uptime requirements can require a 99.999% uptime with no more than 6 minutes of downtime per year[3]. These mission-critical systems are often used in emergency response to support emergency workers to safely and quickly reach and provide the required assistance to those in need. As such, mission-critical telecommunications require effective methods to help ensure the system's stability can meet the performance required by its applications. For real-time applications, downtime also includes periods when performance is degraded beyond the levels of its service requirements. Due to the limited downtime or anomalous instances, it would be more efficient to develop a method generating simulated anomalous data than trying to collect anomalous instances from active systems. Additionally, by providing autonomous monitoring for each virtualized system, it can reduce the complexity of applying self-healing methods since fixes can be applied without affecting the larger system [16].

The use of ML to detect faults such as intrusion, DDOS attacks, memory or hardware faults is an area of focus in many research efforts today [14, 25]. Another application of ML models is that they can assist in maintaining the uptime in mission-critical systems by training a model that can detect if a system's performance has degraded beyond acceptable specifications. These specifications change from application to application; however, within call processing, this can be the server's processing time required to set up the communication. This paper aims to create a ML model that accurately detects abnormal performance degradation that exceeds the normal operating parameters of a virtualized system.

Scope. This paper describes a procedure for simulating call processing under different scenarios with varying loads. The call processing setup is two virtual machines loaded with SIPp call traffic with media (SIPp simulates the SIP call and messaging protocol [9]). One virtual machine will be the server, and the other will be the client. The two virtual machines will exist on the same host system, with all communications taking place over the local building network. A definition of degraded performance and how it is classified will be determined by analyzing the normal operating behaviour of the test setup. By defining the specifications of what is expected for the system, a level can be set for what resulting performance on the client side is classified as anomalous or non-anomalous.

Various stressors will be applied to the system using stress-ng (open source software for stressing systems [15]). Each type of stressor shall have two different load levels applied, a low level that will not result in degraded performance on the client side and one that will. The data is classified as anomalous if the round trip time (RTT) reaches anomalous levels due to performance degradation caused by stressors active on the server side. The RTT consists of the client-server-client communication delay and the server processing time and we will assume reliable client-server communication. Thus, increased RTT indicates increased server processing time due to anomalous server behaviour. While other client or design parameters could be used for classification, RTT is selected as a classifier for validation as it is a metric that could be applied to many mission-critical approaches in communication systems. In theory, the methodology of using measurable design parameters for classifying training data could be adapted to other mission-critical systems by using a different set of more relevant parameters.

The ML models are tested and trained with data from measuring key performance indicators (KPI) on the server side during different scenarios, we will build a training and test data set for the ML models. The scenarios being tested are (i) unstressed scenarios when no stressor is active, (ii) non-anomalous stressed scenarios when a stressor is active, but the RTT does not exceed anomalous levels and (iii) anomalous scenarios where the stressors will cause the RTT to exceed anomalous levels. The models will be tested with untrained scenarios where the data from the stressors are exclusively used for testing. The untrained scenarios will validate that the ML models are accurate for other anomalous scenarios and are not overtrained for the stressors that they were trained with.

From the results, seven well-known ML models were compared over various scenarios using the stressors tested. The diverse types of ML models that were selected include support vector machines, tree-based, cluster-based, and probability-based models. In addition, both classification and regression, as well as both supervised and unsupervised methods, are tested to determine if there are any distinctions in their ability to be used for anomaly detection under this methodology and the generality of the approach.

Contributions. Our main contributions are the following:

- 1) We leverage client-server RTT in the call-setup protocol as the main classifier for defining

anomalies and (non-)anomalous scenarios. The novelty of this approach is that it more accurately classifies anomalies compared to the standard classification of data collected from when the system is stressed or unstressed, since low-stress scenarios may still be tolerated by the call-processing system. 2) We generate accurate anomaly detection that can evaluate degraded client-side performance. All tested supervised ML models achieved F1-scores above 0.99 for trained test scenarios excluding Gaussian Naive Bayes. However, many of the supervised models had reduced performance when comparing the machine-learning results with the stressor test results. Random Forest performed the best across all untrained test scenarios, with the lowest result being 0.980.

Outline. Section 2 will provide an overview of the measurement and anomaly generation applications. Section 3 will provide the methodology for the paper. Section 4 will display and discuss the results of the performance for ML models on the tested scenarios. Finally, in section 5, we present related work.

2 Background

Training ML models require applications to measure and store the KPIs being measured on the system. Generating the data for training the model also requires a system that is under load to monitor and another application to create an anomalous state to generate data for training the supervised ML models.

Measurement Tools & Applications. This section will denote what tools will be used to measure the KPI required for anomaly detection. The selected tools are `vmstat`, `iostat`, and `netstat`. `vmstat` is an application can be used to measure various hardware performance metrics on the virtual machine[26]. It provides access to active/inactive memory, context switching, CPU interrupts, and buffer memory usage. `iostat` command can be used on Linux systems to monitor the IO usage of connected devices[10]. `netstat` is a command that can be used to display network related metrics for Linux based systems. `netstat` can display a large amount of detailed information regarding the system[27].

Anomaly Generation Application. We used `stress-ng` to generate anomalous data on the systems to be tested. `stress-ng` is open-source software that can stress a system in numerous ways through its selection of over 280 selectable and customizable stress tests[15]. The tests can be selected to stress targeted pieces of a system. This can be ideal for generating anomalous data for select KPIs to ensure all of the features we are measuring can have anomalous training data that can stand out from a healthy system measurement. Several authors have researched ML using anomalous data generated using `stress-ng` to simulate various instances such as CPU, memory, and disk faults[11, 12, 6, 21, 20].

SIPp. SIPp is a software that can simulate and allow performance testing of SIP protocol[9]. Using SIPp on two different platforms, one can have user agent client (UAC) and user agent server (UAS) communication using different packet types. SIPp can perform operations during a call, such as executing commands or streaming multimedia such as audio. During communication, it can provide detailed statistics such as RTT, call rate, and other call-related statistics. These statistics allow a user to determine the quality of a given communication. We will use SIPp as the method of producing system and network load on our virtual machine and measuring the RTT for all communication during a period.

3 Data Collection Methodology

This section of the paper will outline the steps and methods used for the data collection. The data collection will include the normal and anomalous system data for training the ML models used in this project. The platform used is a VM running using two cores from an Intel Xeon E5-2670, 3GB of memory, and a virtual operating system running Red Hat Enterprise Linux 8.4 (Figure 3).

3.1 SIPp

The call scenario that was used is a customized version of the default user agent client packet capture (UAC PCAP) call scenario offered by SIPp. The PCAP scenario simulates media communication between the client and server end. This media communication generates significantly more load than the standard UAC scenario, which has a small exchange of packets. Additionally, the load can be a better simulation of an actual scenario, as in the UAC PCAP scenario, data is streamed between the two. For the call scenario, a call rate of 70 call/s provided a sufficiently high load on the server

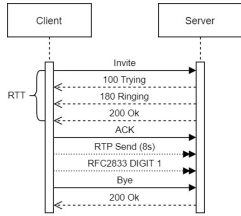


Figure 1: UAC PCAP call scenario

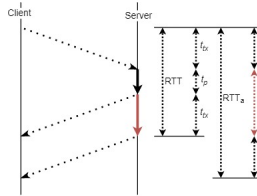


Figure 2: Round trip time (RTT) diagram

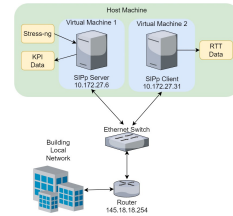


Figure 3: Diagram of the test setup

side but not high enough to cause the server to be overloaded and lead to timeouts due to excess load.

Settings that were adjusted within the default UAC PCAP scenario were the timeout settings that needed to be adjusted to prevent calls from staying open and limiting the startup of new calls. Otherwise, the call rate for the scenario could be unstable. The retransmission time for the invite message was reduced from 500ms to 50ms due to the network’s very short latency. The default 500ms caused large spikes in the average due to the long retransmission delay; reducing the retransmission time to 50ms reduces these spikes but still makes them impactful. The RTT is measured from the initiator’s or client’s end, starting the RTT timer on the invite message and stopping it upon receiving all set-up messages just before the initiator sends the ACK (figure 1).

RTT can measure the quality of communication traffic, especially in mission-critical systems where if RTT spikes are left unaccounted for, they could lead to disruptions. Under the assumption that the network connecting two systems is stable and transmission time (t_{tx}) remains unchanged, changes in RTT can be used to quantify the changes in the time to process (t_p) a message provided to the server before replying (Figure 2). Suppose the connection of a system is stable and the RTT is increased. In that case, the increased RTT is due to increased processing time. This increase in processing time can be due to the server being in an anomalous state (t_{p_a}), increasing the server’s time to respond. Monitoring the server’s KPIs while measuring the RTT, one can classify the RTT from the client’s end as anomalous or non-anomalous and apply that same classification to the KPIs logged on the server end. With this application, one can build enough data to train and test ML models to monitor the KPIs of the server. This claim of a stable network while monitoring a system under a load of SIPp voice calls is similar to other studies that used a similar methodology for assessing the viability of containers [19] or studying anomaly detection [21]. Under stable network conditions data can be collected to train ML models that could be applied to similar active systems where these assumptions aren’t present and RTT cannot be used alone as a suitable metric for determining system stability.

3.2 Anomalous State

A definition for what is classified as anomalous must be made before determining what is anomalous and the acceptable RTT. An anomalous state should have an RTT that exceeds the levels expected for transmission. Unexpectedly high RTT could cause degraded communication quality experienced by the client. With such a definition, ML models can monitor and ensure a system’s reliability.

To determine what value of RTT is anomalous, the RTT of an unstressed system running the UAC PCAP scenario is measured for 20 hours. The RTT is measured from the client’s end to evaluate the quality of the communication from the client’s perspective. Meanwhile, the KPIs are measured on the server end. The RTT is averaged over the time frame of the data point in the KPI log so that the averaged RTT can be used to classify the data point. Mathematically, the classification for what is outside normal distribution is any RTT that exceeds the third standard deviation for the averaged RTT data points over the full 20-hour measurement time period.

To measure the effectiveness of the ML models, the metrics that are used are accuracy (eqn. 1) and F1 score. (eqn. 4). To perform these calculations, the classification matrices need to be defined for the provided test scenarios that are performed. For any tests where the expected RTT level is below the anomalous RTT level, a positive result is the model correctly predicting a non-anomalous result. For any tests where the expected RTT is above or equal to the anomalous RTT level, a positive result is the model correctly predicting the system’s state as anomalous. The detailed classification matrices are in the appendix 6.1.

3.3 Stress test selection

The following tests were selected for creating data that will stress the measured KPI of the system and create both anomalous and non-anomalous data for different segments of the system. The collected data from these tests are used for training and testing of the models:

- **CPU** to cause CPU load, stress-ng iterates through a list of loads such as floating point calculations, square root and other computational loads such as the eight queens problem used in computational load tests
- **icache** to simulate a load causing interference in the instruction cache leading to further instruction stalls and delays to the pipeline
- **aio** to cause stress on the io by triggering several small writes and reads to the disk
- **UDP** to cause stress on the local network sockets for UDP
- **rawsock** to create additional stress on the TCP/IP using the raw sockets on the local host

In addition to these stress methods, the four additional stressors are selected. These stressors will affect similar KPIs but will strictly be used to test the ML models and not to train them. With this data, the anomaly detection accuracy can be tested for other similar stressing scenarios. These will test for overtraining and ensure that models can classify any similar stimuli that may cause the system to enter an anomalous state. The chosen untrained stressors are:

- **matrix** this iterates through several matrix operations. By enabling the yx option, the stressor can cause both CPU and cache interference
- **revio** has workers writing to temporary files on the hard disk. The writes are performed in reverse order. This is similar stress usage to aio; however, revio only performs writes
- **rawudp** similar to rawsock, but instead it will stress using UDP
- **rawpkt** sends and receives packets over the localhosts ethernet port

The provided tests are completed using different parameters to allow them to create enough stress to achieve anomalous RTT, as well as create a low-stress version that will have a non-anomalous RTT when performing non-anomalous data generation. The universal parameters for each stressor are the number of workers, and runtime and deadline parameters can be adjusted to cause different load levels. There are a few other stressor-specific parameters that are changed, such as enabling the yx option in the matrix stressor to cause cache interference.

The resulting data from testing shall have at least 97% of the generated data points as anomalous or non-anomalous depending on whether it is an anomalous test or non-anomalous test. The amount of variation in the average RTT for each test is different due to the different stressors. However, for non-anomalous stress loads, the RTT shall also be below the anomalous RTT threshold.

Other stress test options from stress-ng were tested. However, among the tests selected, they were also selected for being reproducible by producing RTT above the anomalous threshold consistently.

4 Evaluation

We present the results from data collection and methodology described in Section 3. We present the test results from determining normal RTT with the applied system, the application of the ML models, analysis of the applied stress tests and how the ML models applied to the system react to the stressors.

4.1 Determining Normal Conditions

The calculation of the anomalous threshold was completed by collecting over 20 hours of system KPI and RTT data while SIPp was active. A call rate of 70/s was selected since it produced a relatively stable connection while generating enough load on the system. The 20 hours of test data were collected in two 10-hour intervals, with the systems being fully reset between the test intervals.

The chosen frame time for the logging was 6 seconds, allowing enough time to execute the 2s monitoring intervals required for both `vmstat` and `iostat` as well as having minimal impact on the

Table 1: RTT metrics from unstressed system runs

Unstressed RTT Metrics			
Metric	1st run	2nd run	Overall
Average over full duration	4.689ms	4.662ms	4.675ms
Standard deviation using rtt	1.991ms	1.892ms	1.942ms
3rd standard deviation using rtt	10.663ms	10.339ms	10.503ms
Standard deviation using rtt for the frame	1.339ms	1.372ms	1.355ms
3rd standard deviation using rtt for the frame	8.707ms	8.776ms	8.741ms

system KPI. The received RTT data was then parsed and averaged over the 6s logging period so each logging frame would have a correlated averaged RTT. A shorter logging interval could be selected if a mission-critical application required a quicker reaction time to an anomalous state, however, this would lead to a larger impact on system performance which could be an issue for systems with limited hardware.

From the measured results, the majority of the RTT ended up being less than 25ms, with averages over the frame being between 4-5ms. Some of the spikes in RTT could be explained by network congestion, as the tests are not performed on a closed network. Due to the bursty nature of RTT there were several measurements exceeding 50ms, however, averaging the RTT for the duration of the frame significantly lowered the variance of RTT (Tbl. 1). The RTT around 50ms are caused by retransmissions where the sender didn't receive the ACK packet within the 50ms timeout and would then resend the packet.

By using the third standard deviation for the RTT of a frame from the client side and the KPIs that were taken during that same 6-second frame interval on the server side, a given KPI for that frame can be classified as normal or anomalous to provide training data for the ML models.

4.2 Applying ML Models

We describe the process of tuning of the parameters of the ML models we used. All collected data was pre-processed using linear scalar without using the mean to center the data before the scaling. The libSVM package offered through Python was used for the support vector machine methods[7]. The polynomial kernel was used for both Support Vector Classifier (SVC) [2] and ν -Support Vector Regression (ν -SVR) [22], while One Class Support Vector Machines (OC-SVM) [24] performed best using the radial basis function (RBF) kernel. Due to the unequal number of non-anomalous cases in the training data, an additional non-anomalous test from the unstressed test case weight calculation was enabled to correct this offset. OC-SVM was trained using only non-anomalous data and was tuned by adjusting the ν parameter to adjust the upper and lower bounds. A balanced approach was chosen to accurately detect anomalous instances while limiting the number of false positives occurring during non-anomalous data.

The Gaussian Naive Bayes (GNB) [29], k-Nearest Neighbors (kNN) [5], Decision Tree (DT) [23], and Random Forest (RF) [2] models were all applied using crates (compilation units) available in Rust programming language. GNB and DT are available through the Linfa-Trees and Linfa-Bayes crates(v.0.6.0), while kNN and RF are available in the Smartcore crate (v.0.2.1) [1]. The GNB model was tuned by adjusting the variance smoothing, the variance smoothing was increased from the default to add a portion of the feature with the largest variance to the stability equation. When training the kNN model, the k parameter for selecting the number of clusters was tested with k values between 3 and 25. The number of clusters was tested using Euclidian, Hamming, Manhattan, and Minkowski distance metrics and found that Manhattan performed the best. The cover tree search algorithm was selected due to its increased speed and efficiency, it also improved accuracy compared to the brute force approach of Linear Search.

Tuning the tree-based models required adjusting the parameters for the split decision-making criteria. Entropy and Gini splitting criteria are available with these rust crates. Gini impurity measures the probability of a data point being misclassified if inserted into a randomly selected point. Entropy measures the disorder or variable composition of a provided node. A split is decided based on which split will decrease the entropy the most and thus increase the level of information the split provides. Both were tested on DT and RF, DT benefited more from the Entropy criterion while RF performance increased with Gini splitting. Both RF and DT had decision making parameters tuned to adjust the weight required to make a split. The default weights were low given the size of our dataset and would result in specialized nodes and trees with larger depths that proved impractical, leading to misclassifying some of the test data.

Table 2: Prediction accuracy results for individual trained tests

Test	GNB		ν -SVR		OC-SVM		SVC		kNN		DT		RF	
overall	83.25		99.14		79.93		99.31		99.31		99.03		99.27	
unstressed	98.49		99.33		77.48		99.16		99.33		99.16		99.33	
RTT Level	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low
aio	98.95	96.84	98.95	98.74	98.95	68.2	98.95	98.74	98.95	98.74	98.95	98.11	98.95	98.74
cpu	100.0	58.11	100.0	99.58	77.41	86.95	99.81	99.58	100.0	99.58	99.81	98.32	100.0	98.95
icache	56.84	93.41	97.47	99.34	25.89	96.70	100.0	99.34	99.79	99.12	99.37	98.68	100.0	99.34
rawsock	100.0	90.30	100.0	97.23	99.59	90.89	100.0	97.03	100.0	97.03	100.0	97.03	100.0	96.83
udp	100.0	19.2	100.0	99.8	100.0	58.8	100.0	99.8	100.0	99.8	100.0	99.8	100.0	99.8

4.3 Model Results

Overall, all models produced an F1 score above 0.8 for unstressed, anomalous, and non-anomalous stressed scenarios. With ν -SVR, SVC, kNN, DT, and RF producing F1-scores scores above 0.99. The RF model has the best F1-score for anomalous scenarios resulting in an overall F1 Score of 0.99898. ν -SVR, kNN, and RF are tied for best overall F1 score for unstressed scenarios with a score of 0.99663 (Fig. 4). In comparison, ν -SVR performs the best in non-anomalous with an F1 score of 0.99663. SVC achieved the highest accuracy overall trained cases despite not having the highest F1 score in the three test scenarios types.

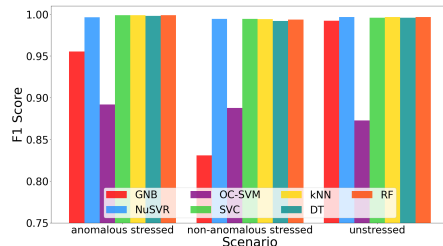


Figure 4: F1 score of models for different test scenarios

As OC-SVM is unsupervised learning, it was unable to reach the same level of accuracy compared to supervised training methods. It obtained F1 scores of between 0.87287 and 0.89189 for anomalous, non-anomalous and unstressed scenarios (Fig. 4). While having generally lower scores than the other models, OC-SVM did outperform the GNB model during several low-stress test cases.

For a model to be accurate in classifying the state of a system, it is essential to balance classifying both anomalous and non-anomalous scenarios. An example of an improper balance can be seen in the GNB and the OC-SVM models when looking at the icache stressor scenario, both have an accuracy of below 60%, but their accuracy in low_icache is above their overall accuracy (Tbl. 2). In such cases, the boundary between anomalous and non-anomalous states is more within the non-anomalous space resulting in numerous misclassifications.

Test cases with unstable bursty behaviour regarding RTT such as aio and rawsock had lower accuracy due to creating more anomalous points within the non-anomalous stress tests. Models had difficulty classifying these RTT bursts often resulting in misclassifications during the bursts. There are several possible reasons for these misclassification. One possible cause is that there is not enough variance in classifying RTT values in the training data. Depending on the test, there could be a lack of data points closer to the RTT threshold resulting in some misclassifications of near threshold values. An improvement that could be tested is to include data sets that vary around the threshold between anomalous and non-anomalous. This could improve the accuracy of these intermediate results, but it could also lead to more misclassifications of non-anomalous data and more false alarms. The second is that some stressors caused a bursty nature in the RTT, which may be challenging to detect, provided the current KPI. A shorter logging interval may allow for detecting these short bursts at the expenditure of higher computational cost. However, there is an argument that, depending on the system, short bursts of high RTT should not impact the overall system’s stability. Furthermore, detecting anomalies that sustainably impact the systems should be the higher priority.

4.4 Untrained Results

The untrained results were obtained with the same test setup and method with 1000 test data points for each anomalous and non-anomalous scenario. The results showed many ML models could not classify either the anomalous or non-anomalous case for a given stressor. Both SVC and ν -SVR had reduced F1-scores for all untrained stressors. The other models attained performance similar to their trained counterparts for matrix and revio test cases. The networking stressors were more challenging with DT, RF, OC-SVM, and GNB achieving good performance in rawpkt. However, RF was the only model to perform well in the rawudp scenario. Across all the untrained tests, RF achieved an F1-Score of above 0.9 (Fig. 5) with better accuracy regarding CPU, cache, and io interference and lower performance for the network stressors (Tbl. 3).

Table 3: Prediction accuracy results for individual unknown tests

Test	GNB		ν -SVR		OC-SVM		SVC		kNN		DT		RF	
	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low
matrix	99.3	97.4	100.0	39.0	55.4	94.8	99.9	15.7	100.0	100.0	99.2	100.0	99.9	100.0
revio	94.9	97.7	100.0	0.2	61.8	86.1	100.0	71.8	100.0	99.9	83.4	99.7	99.8	92.0
rawudp	0.0	100.0	0.0	100.0	100.0	0.6	100.0	0.0	100.0	0.0	99.8	16.2	100.0	92.0
rawpkt	91.6	91.7	99.9	2.0	100.0	95.8	99.9	2.0	100.0	2.0	100.0	97.4	100.0	97.6

The decrease in accuracy for both SVC and ν -SVR could be caused by over-training. For both SVC and ν -SVR, a potential cause of over-training is a high C value creating very thin margins[2], out of the two models only SVC has a high C value. However, when testing with lower C values the accuracy was reduced in accuracy overall trained scenarios and the matrix and revio test cases. Revio eventually regains accuracy once reducing the C value significantly. However, the matrix scenario then reduces in accuracy below 1%. Another possible cause of overtraining is that higher degree polynomial kernels can cause overfitting as the shape of the support vector will mirror changes that could be caused by noise rather than fitting the overall trend of the training data [17]. The applied ν -SVR does have an exponential of 7 on the polynomial kernel. However, upon testing lowering the exponential resulted in decreased accuracy for both trained and untrained CPU and io stressors with no effect on the network stressors. The kNN model also had poor accuracy for the tested untrained networking stressors. When performing testing the different methods, the use of Hamming distance metric greatly improves the accuracy of rawpkt and rawudp. However, it also lowers the accuracy of all other trained and untrained test scenarios with an overall accuracy of 91.316% with higher accuracy towards non-anomalous scenarios. Switching the search algorithm to linear search had a similar impact but at an increased computation cost. Moreover, GNB and OC-SVM generally achieved comparatively higher scores despite having lower accuracy in the trained results because these models offer more flexibility at the cost of trained accuracy.

A possible cause for the lower performance of SVC, ν -SVR, and kNN is that the training network stressor scenarios may have different patterns in the KPI values that cause the untrained scenarios to land on one side of the support vector. For kNN, this would instead be caused by the majority of nearest data points within the training data set being classified as anomalous. One possible way to improve these models would be to perform additional training cases covering an improved spectrum of anomalous and non-anomalous scenarios for network stressors. RF achieved high accuracy in trained and untrained results without any overtraining regarding any of the test cases. In comparison, other models needed to give up accuracy as a trade-off for better generalization.

4.5 Discussion

The use of RTT as means of classifying training and test data is a different approach than what is presented in other ML papers that use simulated data [13, 6, 11, 12, 20, 25]. Often, the metric for anomalous is whether or not the stressor is active which for practical applications could require more verification on whether the stressor is causing the system to enter an anomalous state by exceeding the specification of the system. It could be that the system can still handle the additional load of the stressor while maintaining availability. The use of averaging and the third standard deviation of the RTT allows users to define anomalous under the perspective of meeting the system specifications and maintaining availability. From the trained results, this proved to be a viable method as all the models could achieve F1 scores above 0.8 for each scenario.

Analyzing the provided results from the trained scenarios shows that many of the ML models achieve a stable F1 score across all scenarios, excluding GNB, which has the lowest F1 score for non-anomalous despite achieving an F1 score greater than 0.95 for both anomalous and unstressed (Fig. 4). Among OC-SVM and GNB, there is a concern for causing false alarms for non-anomalous scenarios with higher UDP stress or detecting anomalous stress within the instruction cache as these tests had lower accuracy (Tbl. 2). For the other support vector machine, kNN and tree-based models did not display significant levels of lower accuracy for a particular test case in the trained results. They achieved F1-scores above 0.99 for all trained scenarios (Fig. 4).

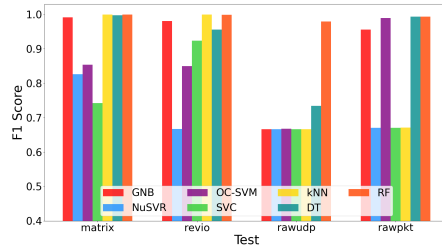


Figure 5: F1 score of models against untrained test scenarios

The untrained results presented a different perspective. They tested how well the ML models could generalize the trained results and allow the accurate detection of anomalous RTT for untrained scenarios. Out of the tested ML models, only RF achieved F1 scores above 0.97 for all test cases (Fig. 5). GNB, DT, and OC-SVM also have notable performances, as each showed lower performance only in the rawudp test case. ν -SVR and SVC showed reduced performance across all test cases, while kNN only showed reduced accuracy during the networking test cases. From the untrained results, kNN, ν -SVR, and SVC showed a greater performance decrease despite their high accuracy across the trained test cases, they proved insufficient in generalizing the approach to detecting anomalous RTT.

The resulting applied methods show that RF will achieve the best results unless the ML is trained using anomalous training data that includes all possible anomalous cases. Using other supervised ML with highly trained accuracy could be worth creating an ensemble method with other ML methods such as DT, GNB, and OC-SVM. These models showed better performance for generalizing the trained results to create an accurate model that could detect when the systems state and provide improved assistance in maintaining availability. However, a more generalized and easy to train model could still be advantageous for self-learning applications where the model needs to completely retrain itself, which can be time-consuming and computationally intensive for large datasets.

The results show that when training ML models using this methodology it is best to use a tree-based model or pair more generalized models with other ML when not able to simulate all possible loads when accuracy is of greater importance. If applying these to a telecommunication system with limited computational resources to adjust the training parameters of models such as reducing the possible depth for tree-based models or increasing the polling interval from 6s if accuracy is more important than time response.

5 Related Work

There is a growing interest in developing automated monitoring and anomaly detection for various computer systems. Many focus on different networking levels, such as core routers[13] and network function virtualization[21]. Other papers focus on cellular networks evaluated ML techniques using data collected from active systems with anomalous data that human personnel has evaluated [6, 8]. These applications do not apply to mission-critical systems and have anomalous events lasting hours and enough data to contribute to the training process.

Regarding virtualized environments, several papers evaluate the application of anomaly detection within docker containers, and virtual machines [11, 21]. Their application included a real-time running application that was also evaluated regarding response time. Other papers focus on modularized systems such as cluster or cloud-based services [25, 12, 20]. These applications offer metric measuring from individual virtual machines that offer modular monitoring. However, the application differs by using the data to determine whether the whole cluster or cloud is anomalous rather than a single virtualized instance. One of the papers has a similar method of training using a low and high-stress application, increasing their models' accuracy. Although it differs in that both stress levels were classified as anomalous [25].

The approach to anomaly classification differs depending on the ML application. Several papers seek to simulate hardware failures and classify anomalous as the stressor being applied[11, 6, 20]. In this paper, the approach to anomalous classification is unique in that it is instead classified by the resulting impact on the system and whether it degrades the performance beyond the specified RTT.

6 Conclusion

This paper proposed a method of training ML models that can detect when given server-client communication exceeds the expected bounds by measuring only the KPI of the server side. Although different models proved to have different performance levels at detecting whether the system was in an anomalous state when provided with KPI metrics from a system loaded with a SIPp UAC with media simulations. With the provided training and test data, out of the seven models, random forests proved to be the best method to measure both the tested trained and similar untrained scenarios accurately. The evaluated methodology can provide the potential for generating anomaly detection, ensuring a more robust and reliable system for many applications and providing additional assurance for mission-critical applications. For future work, a suggested action would be to explore the accuracy of non-binary classification in order to provide improved feedback from anomaly detection. Further

testing could be done to discern the most impactful KPI for training and to validate other possible classification parameters than RTT.

References

- [1] <https://crates.io/>: Rust Package Registry (Jun 2022)
- [2] Aurelien, G.: Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media (Mar 2017)
- [3] Avdotin, E., Bankov, D., Khorov, E., Lyakhov, A.: OFDMA Resource Allocation for Real-Time Applications in IEEE 802.11ax Networks. In: 2019 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom). pp. 1–3 (Jun 2019). <https://doi.org/10.1109/BlackSeaCom.2019.8812774>
- [4] Azizi, S., Zandsalimi, M., Li, D.: An energy-efficient algorithm for virtual machine placement optimization in cloud data centers. *Cluster Computing* **23**(4), 3421–3434 (Dec 2020). <https://doi.org/10.1007/s10586-020-03096-0>
- [5] Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge University Press, 1 edn. (Jun 2012). <https://doi.org/10.1017/CBO9780511804779>
- [6] Casas, P., Fiadino, P., D'Alconzo, A.: Machine-Learning Based Approaches for Anomaly Detection and Classification in Cellular Networks. In: TMA (Apr 2016)
- [7] Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* **2**, 27:1–27:27 (2011)
- [8] Ciocarlie, G.F., Lindqvist, U., Nováczki, S., Sanneck, H.: Detecting anomalies in cellular networks using an ensemble method. In: Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013). pp. 171–174 (Oct 2013). <https://doi.org/10.1109/CNSM.2013.6727831>
- [9] Gayraud, R., Jacques, O., Robert, D., Charles, W.: SIPp (Apr 2014), <http://sipp.sourceforge.net/doc/reference.html>
- [10] Godard, S.: iostat(1) - Linux man
- [11] Gulenko, A., Schmidt, F., Acker, A., Wallschläger, M., Kao, O., Liu, F.: Detecting Anomalous Behavior of Black-Box Services Modeled with Distance-Based Online Clustering. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). pp. 912–915 (Jul 2018). <https://doi.org/10.1109/CLOUD.2018.00134>
- [12] Gulenko, A., Wallschläger, M., Schmidt, F., Kao, O., Liu, F.: Evaluating machine learning algorithms for anomaly detection in clouds. In: 2016 IEEE International Conference on Big Data (Big Data). pp. 2716–2721 (Dec 2016). <https://doi.org/10.1109/BigData.2016.7840917>
- [13] Jin, S., Zhang, Z., Chakrabarty, K., Gu, X.: Accurate anomaly detection using correlation-based time-series analysis in a core router system. In: 2016 IEEE International Test Conference (ITC). pp. 1–10 (Nov 2016). <https://doi.org/10.1109/TEST.2016.7805836>
- [14] Khalil, K., Eldash, O., Kumar, A., Bayoumi, M.: Machine Learning-Based Approach for Hardware Faults Prediction. *IEEE Transactions on Circuits and Systems I* **67**(11), 3880–3892 (Nov 2020). <https://doi.org/10.1109/TCSI.2020.3010743>
- [15] King, C.I.: stress-ng (stress next generation) (Jul 2022), <https://github.com/ColinIanKing/stress-ng>
- [16] Liu, Y., Zhang, J., Jiang, M., Raymer, D., Strassner, J.: A Case Study: A Model-Based Approach to Retrofit a Network Fault Management System with Self-Healing Functionality. In: 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008). pp. 9–18 (Mar 2008). <https://doi.org/10.1109/ECBS.2008.30>
- [17] Mostafa, Y.A., Magdon-Ismail, M., Lin, H.T.: Learning From Data. AMLBook, 1st edn. (Aug 2017)
- [18] Program, C.N.A. (ed.): Connecting networks. Companion guide. Cisco Press, Indianapolis, Indiana, 1st edition edn. (May 2014)
- [19] Romanov, O., Nesterenko, M., Fesokha, N., Mankivskiy, V.: Evaluation of productivity virtualization technologies of switching equipment telecommunications networks. *Information and Telecommunication Sciences* (1), 53–58 (2020)
- [20] Samir, A., Pahl, C.: Detecting and Predicting Anomalies for Edge Cluster Environments using Hidden Markov Models. In: 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC). pp. 21–28 (Jun 2019). <https://doi.org/10.1109/FMEC.2019.8795337>
- [21] Sauvanaud, C., Lazri, K., Kaâniche, M., Kanoun, K.: Towards Black-Box Anomaly Detection in Virtual Network Functions. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W). pp. 254–257 (Jun 2016). <https://doi.org/10.1109/DSN-W.2016.17>

- [22] Schölkopf, B., Smola, A.J., Williamson, R.C., Bartlett, P.L.: New Support Vector Algorithms. *Neural Computation* **12**(5), 1207–1245 (May 2000). <https://doi.org/10.1162/089976600300015565>
- [23] Segaran, T.: *Programming Collective intelligence*. O’Reily Media (Aug 2007)
- [24] Tao, C., Li, T., Huang, J.: Kernel Choice in One-Class Support Vector Machines for Novelty and Outlier Detection. In: 2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI). pp. 116–120 (Oct 2020). <https://doi.org/10.1109/MLBDBI51377.2020.00026>
- [25] Tuncer, O., Ates, E., Zhang, Y., Turk, A., Brandt, J., Leung, V.J., Egele, M., Coskun, A.K.: Online Diagnosis of Performance Variation in HPC Systems Using Machine Learning. *IEEE Transactions on Parallel and Distributed Systems* **30**(4), 883–896 (Apr 2019). <https://doi.org/10.1109/TPDS.2018.2870403>
- [26] Ware, H., Frederick, F.: `vmstat(8)`: Report virtual memory statistics - Linux man
- [27] Welsh, M., Cox, A., Hoang, T., Eckenfels, B.: `netstat(8)`-Linux man.
- [28] Wong, D., Yip, S.: Machine learning classifies cancer. *Nature* **555**(7697), 446–447 (Mar 2018). <https://doi.org/10.1038/d41586-018-02881-7>
- [29] Zhang, H.: The Optimality of Naive Bayes **2** (Jan 2004)

APPENDIX

6.1 Classification Matrices and Equations

Below are the classification matrices that were used to appropriately classify the collected data from tests as well as calculate the F1-score for determining performance.

True Positive	False Positive
Below Anomalous RTT & Classified as Non-Anomalous False Negative	Below Anomalous RTT & Classified as Anomalous True Negative
Below Anomalous RTT & Classified as Anomalous	Below Anomalous RTT & Classified as Non-Anomalous

Table 4: Anomaly Classification Matrix for Unstressed and Non-Anomalous Tests

True Positive	False Positive
Above Anomalous RTT & Classified as Anomalous False Negative	Above Anomalous RTT & Classified as Non-Anomalous True Negative
Below Anomalous RTT & Classified as Anomalous	Below Anomalous RTT & Classified as Non-Anomalous

Table 5: Anomaly Classification Matrix for Anomalous Tests

$$Accuracy = \text{positive} / (\text{positive} + \text{negative}) \quad (1)$$

$$Precision = \text{true positive} / (\text{true positive} + \text{false positive}) \quad (2)$$

$$Recall = \text{true positive} / (\text{true positive} + \text{false negative}) \quad (3)$$

$$F1 \text{ Score} = 2 \frac{Recall \times Precision}{Recall + Precision} \quad (4)$$