Practical Performance Degradation Mitigation Solution using Anomaly Detection for Carrier-Grade Software Networks

Marius Corici[1], Teodora Sandra Buda[2], Ranjan Shrestha[1], Eleonora Cau[1], Taner Metin[1], Haytham Assem[2]
[1]: Fraunhofer FOKUS Institute, Berlin, Germany,
{marius-iulian.corici; ranjan.shrestha, eleonora.cau, taner.metin}@fokus.fraunhofer.de
[2]: IBM Ireland, Dublin, Ireland, {tbuda, haythama}@ie.ibm.com

One of the critical technologies required for the large scale acceptance of network functions virtualization within carrier-grade communication systems is the maintenance of a predictable performance level for the software network functions. However, due to the specific resource allocation and to the implementation of the software itself, the performance of the network functions tends to degrade in time, thus, reducing the capacity of the system to serve the specific service requirements. This article introduces a practical solution for performance degradation detection and mitigation for telecom oriented software networks. Also. it includes the mechanisms to interact with the software network and a mechanism for abnormal behaviour detection, as basis for the performance degradation automatic decisions. Furthermore, the solution is exemplified as an addition to the current 5G core network architecture and evaluated on a testbed based on the Fraunhofer FOKUS Open5GCore toolkit and IBM DeepAD. The measured results show that such a solution is feasible and should be further investigated to be integrated in the next generation carrier-grade software networks management for enabling an autonomous long duration functioning of the network.

Keywords— software networks, anomaly detection, machine learning, carrier grade 5G systems

I. INTRODUCTION

Currently, there is a wide acceptance of software network functions and networks to be deployed on top of cloud infrastructures. Building on these developments, the Network Functions Virtualization (NFV) initiative developed a set of management features able to satisfy the requirements of the carrier grade telecom networks, especially concentrating on the deployment and configuration of the software network functions and on the runtime adaptation towards a scalable and stable system.

With this functionality, full virtual networks can be deployed in parallel, on top of the same cloud infrastructures in a matter of minutes, customized and configured to address the specific requirements of the services.

To fulfill the goal of carrier-grade systems, the management of these virtual networks should be able to monitor the status of the network and to adapt accordingly. This functionality is currently missing from the NFV standardization and initial de-facto platforms.

Especially important is the maintenance of the specific level of performance for a large duration of time and for a large number of subscribers, while serving different momentary loads, as it is the case of the current physical networks.

However, software networks are prone to the degradation of performance in time, due to side effects from the programming and due to the cross-layer interaction with the underlying cloud system. Because of the long time duration in which this performance degradation happens and due to the acceptance of flexible load levels through scaling procedures, such events are not noticeable to the current monitoring systems. Their effect is mainly to reduce the quality of the service offered to the subscribers and increase the usage of the network resources.

In this paper, we propose a comprehensive management solution to monitor, detect and mitigate the performance degradation within software telecom networks. The solution acts as an intelligent, extended "watch-dog" for the software components, determining when an anomaly in behaviour is happening, compared to an expected level of performance. As it is foreseen that parallel security solutions will be deployed, we assume that the performance degradation is due to the behaviour of the network functions, including faults in the software network functionality, and thus require a specific mitigation solution separated from misbehavior due to attacks.

The main contribution of this paper is the practical implementation of such an end-to-end system, acting as an initial best practice for the implementation of such features within software networks. Instead of concentrating on the development of new algorithms, the solution aims to provide the mechanisms through which such functionality can be implemented on top of cloud networks, through this opening the door for optimized intelligence within the management system.

The solution is exemplified using the Fraunhofer FOKUS Open5GCore toolkit [5], as a standard-based, prototype implementation of a 5G system and the IBM DeepAD [7] as an evolved solution for anomaly detection using machine learning techniques.

The two systems were combined into a comprehensive system in which the management of the carrier grade system interacts locally or remotely with the machine learning system for the transmission of monitoring information and for receiving insight when a performance degradation happens.

The remainder of the paper is organized as follows: Section II provides an overview of the background technologies involved in the solution. Section III describes the concept and the architecture of the proposed solution. Section IV describes the practical implementation, while Section V presents the experimentation results and their assessment and in Section VI the conclusions are provided.

II. PROBLEM STATEMENT

A first step of the deployment of the carrier-grade networks was the porting of the existing architectures for physical components to the cloud environment. In case of the 3GPP Evolved Packet Core and its evolution within the 5G Core Network, this included the porting of the control plane entities (represented by the Mobility Management Element – MME), the subscriber data bases (represented by the Home Subscriber Server – HSS) and the data plane entities (represented by the Serving and Packed Data Network Gateways – S-GW/P-GW).

With the initial tests and deployments immediately it was observed that the network functions behaviour highly differs from one cloud environment to another, leading the software providers to require a large amount of testing as to be able to reduce the uncertainty of their software misbehaving.

Additionally, as the resources were dynamically allocated on top of cloud infrastructures, the behaviour of such a system was prone to a large amount of side-effects from the parallel deployments and from the underlying resource allocation environment. Seen from the software network functions, such effects are drastically changing the performance of the end-to-end system.

Same effects could be seen from the mismatching between the planned resources scaling and the actual usage of the network.

Furthermore, a software component may malfunction due to hardware failure or occurrence of anomalies in allotted resources at software level. The anomalies may vary from known to unknown types that may have been triggered from a large set of parameters or coming from the previously mentioned side effects.

To mitigate all of these at once, the current alternative is to reboot the complete system at given time intervals. Due to the high level of reliability expected (99,999% of time availability), such a reboot requires a large amount of resources, as it implies the need of a standby system to take over during the reboot [8].

A better alternative to this would be the detection of independent anomalies detected per component basis. This helps to find out which component is actually affected and which parameter actually triggering the anomaly in the system.

Furthermore, in these complex systems where anomaly can occur from any point, which can sometimes be hard to trace, an anomaly detection based only on static thresholds may not be accurate, as the usage of the resources is highly dependent on the system usage, which is specific to a given deployed service.

Instead, a more dynamic solution is required, able first to determine which is the normal usage for the specific deployed service and then to determine when the normal usage is consuming more resources than expected would fit better [1].

III. CONCEPT AND ARCHITECTURE

The sudden irregularities in system parameters of the components may degrade the performance of the overall system severely affecting its reliability and end-to-end services. These anomalies or unusual data patterns which do not conform to expected behavior may be caused by system malfunction, change in network resources demand etc. The ability of the system deployed in any cloud environment must be able to offer and maintain minimum QoS defined by set of service KPIs in dynamic environment which is the measure of the reliability of the system. The overall concept is to maintain the reliability by engaging anomaly detection based Machine Learning (ML) algorithms on top of the NFV architecture. The role of machine learning is to extend currently available fault management system in NFV orchestration to detect more foreseen/unforeseen anomalous events in dynamic environments and predict more accurate actions in order to maintain high availability of the system and keep it operational.

The anomaly detection based ML algorithms has various application domains such as fraud detection, health, network management etc which should be able to act proactively and generate actions before the failure of the monitored components.

The proposed performance degradation detection and mitigation solution is driven towards providing additional feature improvements in the existing infrastructure.

- Based on the logged metric data, it is able to cluster different types of features that enables the grouping of specific components in several equivalence classes such as network usage or subscriber clustering.
- The anomaly refers to the abnormal data patterns which can occur from subscribers' level or malfunctioning in the network functions within the system. Such patterns to be detected on time to take proactive actions to main system's reliability.
- The undesired faults may occur which the system can recognize as the variation in resource intake by the network functions and indicate orchestrator to take appropriate actions.
- The monitoring of the overall health of the system is essential for all the network functions. The logging of the metrics in the components can be done varying from small to large intervals of time depending on their nature.

The proposed solution is adapted from the generic architecture in CogNet [2], which on its own it is built on top of ETSI NFV architecture. The 5G components are deployed based on ETSI NFV architecture in a cloud environment. In order to have this customized and extended framework design, additional components are added mainly for monitoring of the components and running machine learning algorithms to be able to gain insight on the monitored data and to trigger automatic network management decisions [1].

1. Log and Monitoring:

To know the real-time status of the network functions, these components need to be monitored and logged constantly. There are various types of system related and customized metrics that are sent to this component. It stores and maintain the data in its database which are categorized in various topics for easy retrieval. The measurement intervals of metrics from each network function can be configured ranging from a second to hours depending on how critical and closely the component

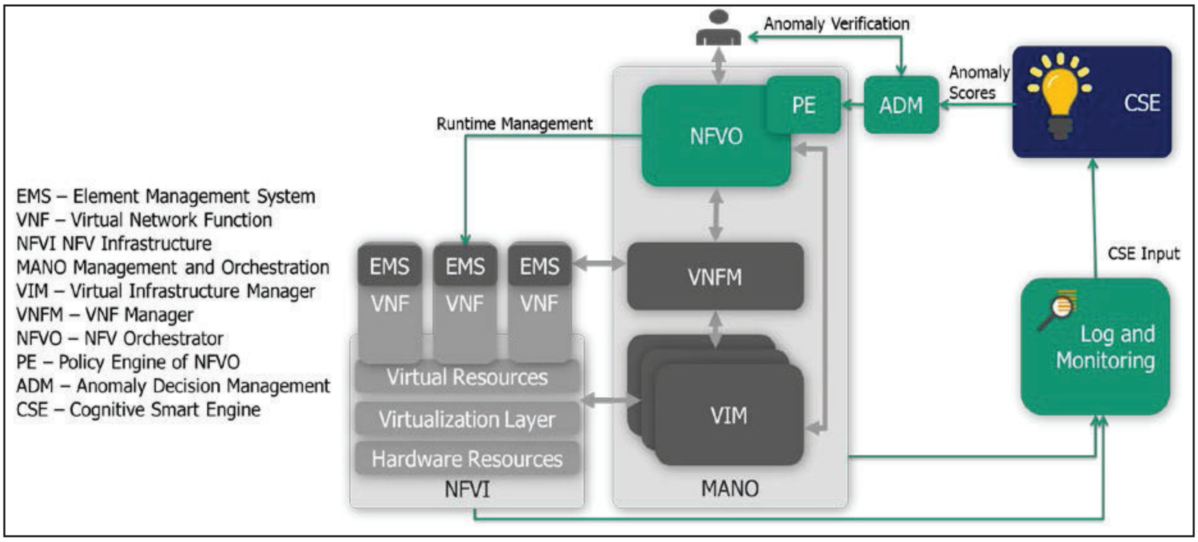

Figure 1 – Architecture of the Proposed Solution

needs to be monitored. The data can be viewed in the form of time series and it exposes various APIs so that other component can retrieve and utilize these monitored data.

2. CogNet Smart Engine (CSE)/ Machine Learning Engine:

The machine learning algorithm is deployed in CSE unit which is used to solve the existing problems of system reliability by vigorous learning and understanding using different methodologies based on different datasets and ultimately making intelligent decisions. The datasets from network functions are classified by features which get translated into KPIs of different types (Infrastructural KPIs, Component resource KPIs, Service KPIs, Reliability KPIs etc) by monitoring component that gather huge amount of information which is then forwarded as an input to the ML block. Those KPIs are collected on different interval periods which can further be aggregated which is enough for the ML algorithm to produce sensible results aiming to increase the reliability and high availability of network functions.

As shown in the Figure 1, the output of the CSE is sent to policy engine responsible for translating output result to policies that are further sent to the NFV components for applying appropriate actions. The policy engine consists of following components described briefly.

- The Clustering Decision Management (CDM) unit receives the result from CSE and applies it to the network functions. In subscriber clustering, the subscribers are grouped in different classes that have common behavior which are based on mobility information, resource consumption, and subscription profiles etc. The mobility information depends on subscribers' movement (speed, direction, area of service). The subscribers are also classified based on patterns of resource usages so that resources can be accordingly allocated for better user experience. The subscribers' profile based clustering help to execute customized policies targeting specific cluster unit. The network clustering can be achieved by placing the network function functionalities on the edge of the network which can receive set of allocated resources depending on its usage based on location, time of use etc.
- The Anomaly Detection Management (ADM) has the function to send CSE anomaly detection module to the orchestrator towards NFV components. The anomaly detection can detect known and unknown anomalies. The known and unknown anomalies are both abnormal behaviors where the mitigation procedures are known for the former and for the latter, it hasn't occurred previously.

IV. TESTBED IMPLEMENTATION

The testbed implementation [4] is based on the above mentioned framework which can logically be categorized into different components which run in Ubuntu 14/16 environment. The Open5GCore [5] toolkit is a functioning packet core unit, a reference implementation for carrier grade standard software network deployed in OpenStack environment. The Open5GCore consists of all the required 5G components of which only the important and interesting ones are shown in Figure 2. The packet core is deployed with overprovisioned MMEs: one is active MME and another one is hot standby MME which replicates the states of subscribers with the help of Redis server. The MME-Load Balancer (MME-LB) knows the state of both MMEs and when request comes from subscribers for attachment procedures, it directs the control plane data via the active MME. To have the network resources available despite an anomaly in active MME predicted by anomaly detection engine, MME-LB is able to execute a complex procedure to transfer load to the hot standby MME by making it active MME and bringing up the service in anomalous MME and its status switched to hot standby. The Benchmarking Tool (BT) is basically an emulator for running benchmark tests and is used to excite the system by executing attachment/detachment procedures (generating traffic) which registers large number of subscribers to the system. This action loads the active MME with large number of requests which may trigger anomaly in that component which is of our interest to investigate. This packet core is implemented in

C/C++. In both MMEs, separate python scripts run whose task is to receive the trigger from the orchestration of Network Management System and based on the policy, they stop and start the MME service.

The logging and monitoring of the network functions are performed by Zabbix server. The system level and customized metrics of the network functions are sent to the Zabbix server of which the interesting ones for our experimentation are CPU, memory, network usages. A python script is executed which uses Zabbix APIs to retrieve the CPU metric data from both MMEs. Then, they are sent as a stream on specific Kafka topics to the Kafka server deployed in Common Infrastructure (CI) in another cloud environment. The message that is sent on the Kafka topic is a json object containing the timestamp, hostname, metric name and metric value of the component.

The anomaly detection in CSE (DeepAD) leverages various explicit generalization models, including deep learning models, to learn the normal behaviour of the data and utilizes a dynamic sliding window for determining a dynamic threshold fitted for each time-series under analysis [7]. DeepAD aims to predict the behavior of the system after having the model trained with huge dataset. One of the models employed is Long Short Term Memory (LSTM) which is a type of recurrent neural network that introduces gates which remember or forget information when passed through them. It is quite applicable for time series data when there are some long or unknown gaps between important events. DeepAD discovers anomalies without the need of golden labels through a dynamic threshold approach, while maintaining the highest levels of true anomaly detection, and reducing the number of false positives compared to the best available technique. In the implementation, a comparison is made between the predicted and actual values for each data point and when the squared difference of those data points exceed the threshold, it is marked as an anomaly. The threshold is determined using a dynamic window which is adjusted for each point to contain past rescaled squared errors to ensure the accuracy is highest. The dynamic threshold approach is used to determine anomaly exploiting the squared prediction error between the actual and predicted values without the need of labels. To determine the dynamic threshold, a set of prior squared errors are stored where scaling function is applied to fit those squared errors between 0 and 1.

The data point is considered anomalous if the squared error of the predicted value is higher than ten times the standard deviation of the prior squared scaled errors. The squared error and threshold are dynamic that influence the prediction at every data point change in order to adapt to better accuracy. The initial 50 timestamps are not considered while calculating the standard deviation so as to reduce the number of false positives [3] [7].

The anomaly detection engine in CSE which is also deployed in CI, listening on some specific Kafka topics get triggers and it receives the CPU metrics data of MMEs. The ML algorithms used by DeepAD which are previously trained with thousands of training data is able to predict if there is any anomalous behavior in the real-time data. The anomaly label is marked 1 or 0 depending on if the anomaly is predicted or not by the algorithm. For every set of input to the CSE, an output result is generated in real-time and sent back to the orchestration unit in Network Management System via socket connection. In this unit, a nodejs JavaScript is listening on socket connection that receives the policy. Based on the policy, if there is any anomaly, the active MME receive trigger in the running python script which then stops and re-starts the MME service, else, it is ignored. The MME-LB is able to quickly switch between the MMEs to make hot standby to active state and maintains constant end to end service.

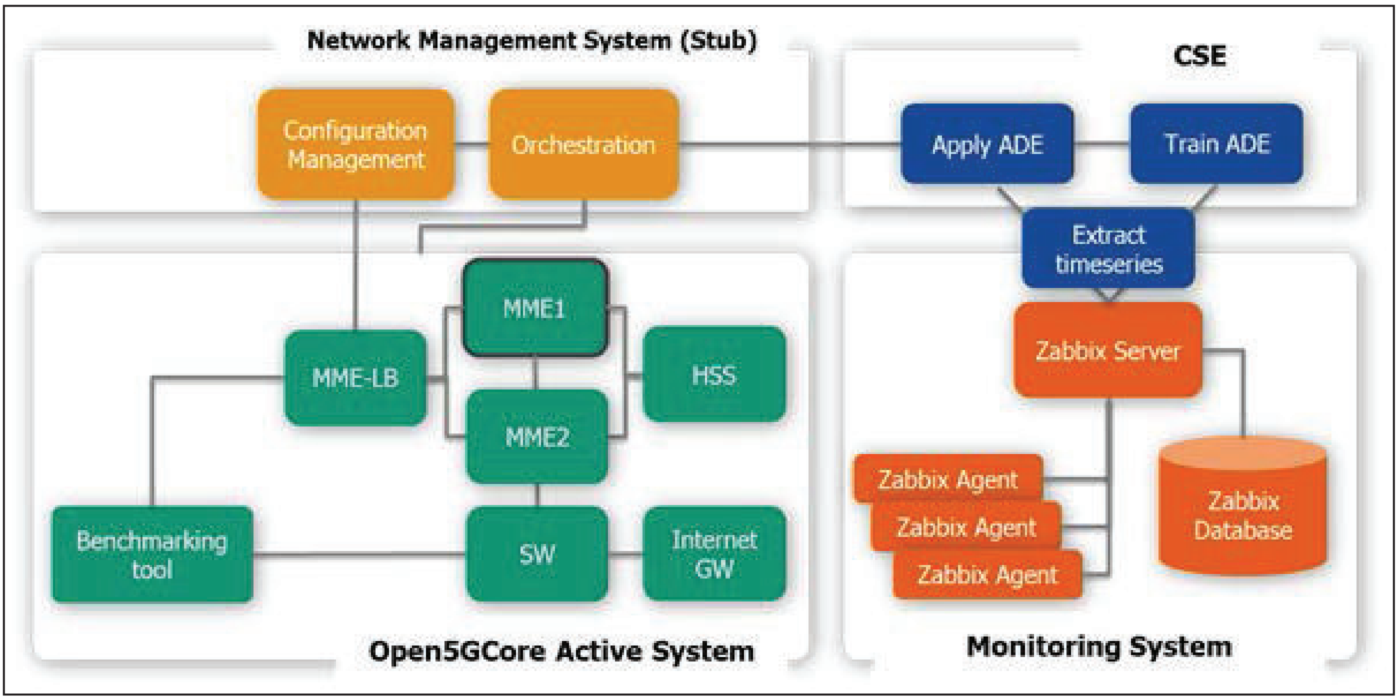

Figure 2 - Testbed implementation for performance degradation mitigation

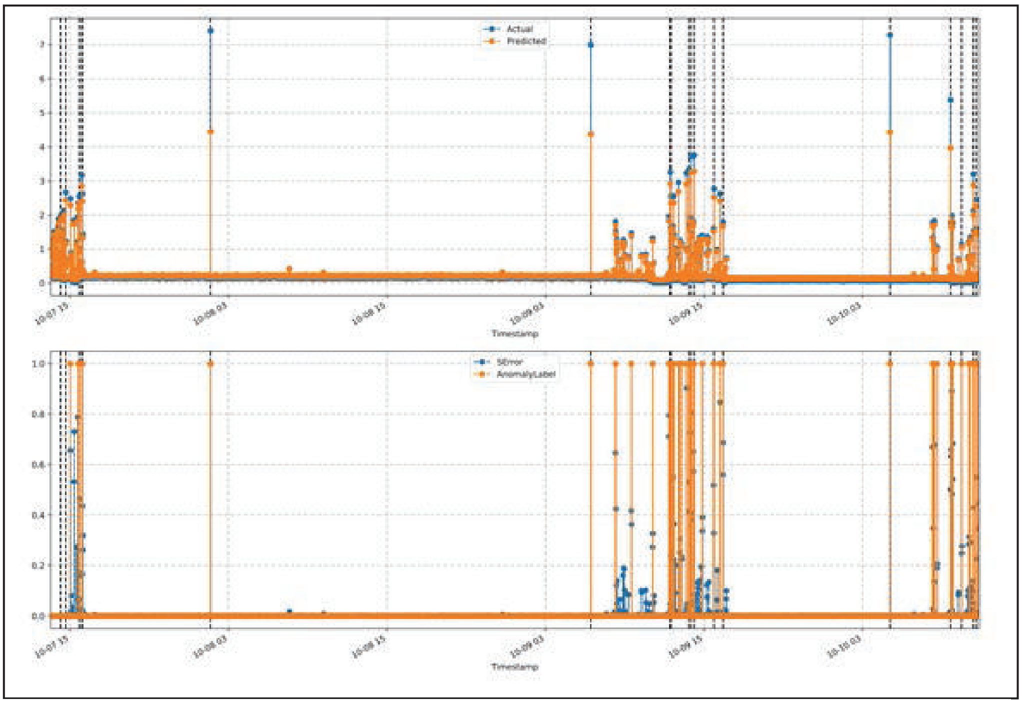

Figure 3 - DeepAD applied to MME1 CPU utilization metric dataset from testbed (Blue – Actual Anomalies, Orange – Predicted Anomalies)

V. EXPERIMENTATION RESULTS EVALUATION

In this experiment, two scenarios are considered. The first one doesn't involve ML component but is set with a fixed threshold for CPU utilization. When the BT component generates traffic by attaching a number of subscribers, the CPU utilization increases and if it exceeds the threshold, a trigger is sent to stop the active MME service. The MME-LB switches to hot standby MME to handle the requests. Another scenario involves CI that includes CSE docker container. The anomaly detection based on LSTM is trained with more than 50000 normal data points. Then, the CPU utilization metric dataset is sent every 5 seconds as data stream to the Kafka server in the CI. The CSE processes the metric dataset and predicts anomaly by setting anomaly label to 1 even before the occurrence of it else anomaly label is set to 0. The evaluation results are categorized as follows [4]:

A. Anomaly detection evaluation results

The anomaly detection module can function in batch or hybrid mode. In batch mode, the model is trained and deployed for scoring on offline collected data from the testbed. In hybrid mode, the model is trained in batch layer but deployed for scoring in near-real time layer. The evaluation parameters such as precision[1], recall[2] and F_1-score[3] are calculated in batch mode based on whether the anomalies detection are within the anomaly window of true anomalies. An anomaly is detected when it occurs within the anomaly window defined in [6] and expressed as 10% of dataset length divided the number of anomalies present. This offers flexibility to detect an anomaly a bit earlier or later than its actual occurrence.

The Figure 3 (first plot) shows predicted versus the actual data points for CPU utilization metric. The predicted data points are generated using the trained model which is based on normal dataset. The Figure 3 (second plot) illustrates the SError representing the deviation between the actual and predicted values along with the AnomalyLabel which is set to 1 upon anomaly occurrence else set to 0 for normal behavior. The dashed lines indicate true anomalies which are labelled during data collection.

The ML module is able to detect all nineteen true anomalies leading to a recall of 1 and from the anomalies reported during data collection phase, two false positives are reported leading to a precision of 0.9 and F_1-score of 0.95. The different measure of errors such as RMSE[4], MAE[5], and R2[6] are calculated to be 0.1426, 0.0636, and 0.8238 respectively which explains low error values and high variance. This shows that the trained model is capable of predicting the output well.

B. Testbed evaluation results

Based on the two scenarios with or without DeepAD as described in above section, the total round trip time (RTT) are calculated for both MMEs as shown in Figure 4. With DeepAD involvement, the RTT is time required for metrics data input to reach CSE through Kafka to the time the response payload containing result is received back. On the other hand, without DeepAD, the metric data input is locally processed. It is observed that the rate of reaction is around

---

[1] Defined as number of true anomalies found to total of anomalies discovered.

[2] Defined as number of true anomalies discovered, out of total number of true anomalies.

[3] Defined as a harmonic mean of precision and recall.

[4] RMSE: Root Mean Square Error, defined as square root of mean of square of deviations between predicted and actual values.

[5] MAE: Mean Absolute Error, defined as difference between mean predicted and actual values.

[6] R2: Coefficient of Determination, defined as the difference between 1 and sum of squares of residuals divided by total sum of squares.

100ms with DeepAD which is quite fast in real time mode to predict an anomaly to take proactive action. Without DeepAD, it is only around 5-7 ms which is obvious as it is handled locally.

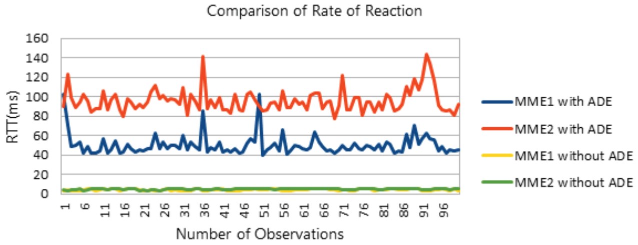

Figure 4 - Rate of reaction measurements using RTT method

The Table 1 illustrates the CPU metrics values of MME1 with DeepAD when number of subscribers are attached to the network with help of BT. The three parameters used for the attachment procedure represents number of subscribers, operations per second (set at 15) and mode of operation (set to 0) which determine the applied load to the system. The average duration represents the total time required to complete the attachment procedure. It is observed that the anomaly label is set to 1 which means the anomaly has occurred for subscribers' attachment from and above 500 where CPU utilization values are higher and possible consistencies are detected by the ML algorithm.

Attachment (with MME1)	Average duration(ms)	CPU utilization	Anomaly Label (1/0)
100 15 0	381.4	0.3574	0
200 15 0	390	0.6046	0
300 15 0	557.5	0.5365	1
400 15 0	574	0.5342	0
500 15 0	540	0.744	1
600 15 0	708	0.8949	1
700 15 0	724.7	0.9944	1
800 15 0	889	1.275	1
900 15 0	925.4	1.8931	1
1000 15 0	942	1.92	1

Table 1 Subscribers' attachment to the network with DeepAD involved

Attachment (with MME1)	Average duration(ms)	CPU utilization	Threshold exceeded(1/0)
100 15 0	600	0.933	0
200 15 0	619.88	1.4693	0
300 15 0	613.8	1.5154	0
400 15 0	639.42	1.6878	0
500 15 0	626.45	1.9207	0
600 15 0	712.4	2.43	0
700 15 0	812.11	2.59	1
800 15 0	889.24	2.78	1
900 15 0	945.8	2.98	1
1000 15 0	1012.4	3.16	1

Table 2 Subscribers' attachment to the network without DeepAD

The Table 2 shows the subscribers' attachment to the network without DeepAD along with the CPU utilization values and threshold set to 2.5. If the threshold is exceeded, it represents an anomaly otherwise normal operation. The threshold value of 2.5 is selected after a number of experiments. But, of course, it can't be better than the prediction made by ML algorithm which is dynamic and based on model which is trained.

VI. CONCLUSIONS

This ML algorithm powered the reliability framework is designed on top of ETSI NFV architecture primarily is able to tackle the problem of sudden increase in workload from external sources and infrastructure level unexpected events such as rapid increase in subscribers' connectivity on the network and failure network functions or malfunction in software components respectively. In order to maintain the minimum QoS level, anomaly detection ML algorithms are proven to be efficient to take proactive actions based on the input metric data. The framework optimizes towards higher availability of network functions and efficient use of network resources leading to the better end to end service reliability of the system. This helps to maintain the overall users' connectivity and network service experience.

As observed in the experimental results, the processing of data points and predictions is done in real-time under 100 ms which allows to take actions quickly compared to the reactive solution which takes around 500 ms for overall switching of the contexts.

It is to be noted that the CSE is running in Common Infrastructure in a cloud environment whereas the network functions are deployed at Fraunhofer FOKUS premises. Thus, the testbed not only showed the feasibility of the solution, but also that the machine learning decision may be centralized and offered as a service by a cloud provider even for deployments which have different locations. Through this, a new market opportunity appears for machine learning based services which could be offered by third parties and not necessarily requiring the embedding into the active system.

Following this key findings, as a next step, the software testbeds offered will include a remote network data analytics function which will remain centralized for a large number of testbeds and it will use anomaly detection for determining that the specific testbeds are appropriately functioning.

ACKNOWLEDGMENT

This work was supported by the EU project CogNet, 671625 (H2020-ICT-2014-2, Research and Innovation action).

REFERENCES

[1] CogNet, deliverable 5.1, Network Security and Resilience Initial Design

[2] CogNet project: http://www.cognet.5g-ppp.eu/

[3] CogNet, deliverable 5.3, High Availability Framework Engineering Release 2

[4] CogNet, deliverable 5.4, Network Resilience Evaluation Framework

[5] Fraunhofer FOKUS Open5GCore toolkit , www.open5GCore.org;

[6] Lavin, Alexander and Ahmad, Subutai. Evaluating Real-Time Anomaly Detection Algorithms -- The Numenta Anomaly Benchmark. 14th IEEE International Conference on Machine Learning and Applications (ICMLA), 2015.

[7] Teodora Sandra Buda, Bora Caglayan, Haytham Assem. DeepAD: A Generic Framework based on Deep Learning for Time Series Anomaly Detection. 22nd Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2018.

[8] R. Natella D. Cotroneo J. Duraes and H. Madeira "On fault representativeness of software fault injection " Software Engineering IEEE Transactions on vol. 24 no. 7 pp. 1