A Streaming Data Anomaly Detection Analytic Engine for Mobile Network Management

MingXue Wang
Network Management Lab
Ericsson, Ireland
mingxue.wang@ericsson.com

Sidath Handurukande
Network Management Lab
Ericsson, Ireland
sidath.handurukande@ericsson.com

***Abstract*—With emerging Network Functions Virtualization (NFV) and Software Defined Networking (SDN) paradigms in Network Management (NM), new network devices and features can immediately become available. Available network resources and services can be altered and optimized in real time to gain the maximum benefit. However, this requires real time analytics information sent to SDN controllers rather than traditional manual offline or batch analytics which deliver outputs in hourly or monthly in NM Systems. As a result, real time analytic is becoming a critical element for NM. In this paper, we describe a configuration free (i.e., non-parametric) streaming data anomaly detection analytic engine for automatic NM system development. We describe the design principles, innovative algorithm design, architecture and implementation of the engine in relation to streaming data and mobile NM. Finally, we present use cases and evaluation results.**

Keywords—Anomaly detection; streaming analytic; machine learning; network management system

I. INTRODUCTION

With emerging NFV and SDN paradigms in NM, new network devices and features can immediately become available. Available network resources and services can be altered and optimized in real time to gain the maximum benefit. However, this requires real time analytics information sent to SDN controllers rather than traditional manual offline or batch analytics which deliver outputs in hourly or monthly in NM systems [1] [2]. Network performance management data is one major data category that is needed to be analysed, and such data is mainly in the form of time series data given the fact the data is continuously generated from the monitoring and management related software components in network nodes, such as base stations. The focus of our work is time series stream data to enable real time insights, i.e., use an anomaly detection analytic engine to identify network abnormal or problematic scenarios in real time for NM use cases. Detected network anomalies can be used to provide insights/discoveries for the network operation engineers, or provide triggers for SDN based network corrective/optimization actions [3] [4].

Mobile network nodes periodically generate various Counter information; the time periods could be, for example, 15 minutes or 5 seconds. These counter values sometimes are used to calculate Key Performance Indicators (KPIs). Abnormalities of these Counters/KPIs often indicate network problems. For example, Attach Request counter defined in telecom standard is about counting number of mobile devices that "register" and "attach" to the mobile network in a given time period [5]. Excessive or abnormal amount of Attach Requests from mobile devices possibly indicate Denial of Service (DoS) attacks [6]. Consequently, many network performance data is closely monitored. However, traditional network performance monitoring use manually set thresholds that are based on a priori knowledge of network. Such static thresholds often do not consider and deal with network dynamics, since network are frequency changed and reconfigured. As a consequence, being able to adapt to changes of network and establish normal behaviour patterns automatically, detecting anomalies of network data in real time is very valuable and significant for mobile NM [7] [8] [9]. When anomalies are detected, either network engineers can be alerted, or automated network/SDN actions can be triggered immediately. Still, existing anomaly detection algorithms [10] [11] are generally offline and batch based, have other integration limitations in NM scenarios (Section II) and they cannot be used for time series data streams for real time analytics. Further related work and limitations are discussed in Section V.

Our work focuses on a streaming data anomaly detection analytic engine for mobile NM system development. Contributions of this paper are as follows:

- The design principles, architecture and implementation details of a streaming data anomaly detection analytic engine that fit to the requirements of mobile telecom NM product development.
- The innovative algorithm design which combines a number of statistical functions in a workflow for time series data stream to produce immediate outputs as data arrives. It is different from related work which does not focus on anomalies in streaming data.
- The implementations are evaluated in context of mobile network use cases to show significance of this work in NM.

The rest paper is structured as follows. Section II describes the engine design principles to overcome limitations of existing systems and the architecture. Section III describes the algorithm design in detail and Section IV presents example use case scenarios and evaluation of the anomaly detection analytic engine. Section V compares and contrasts related work and Section VI presents concluding remarks.

II. STREAM ANALYTIC ENGINE

A. *Engine design principles*

There are many existing machine learning (ML) and analytic frameworks and libraries, such as R packages [12] and MLlib [13]. However, when taken as they are, these libraries do not fully fit requirements of NM system development for streaming data. In the following we discuss some fundamental requirements we identified with respect to NM system development. These requirements are driving factors of our design principles and our motivation to develop a new streaming analytic engine for NM systems.

1) Real-time streaming analytic

Traditionally, analytics workflows in mobile NM systems normally involve collecting data files, parsing data according schemas, storing data in databases or file systems on disks, and then applying OLAP (online analytical processing) to offer analytic reports for subsequent network troubleshooting or optimizations. However, these offline and batch based analytic approaches lack real-time analytic capabilities that can help operators' real time and proactive NM needs.

Unlike traditional batch oriented analytics, real-time streaming analytic is memory based, which does not store data in disks. It is a continuous process and is predictive and proactive in nature. Real-time streaming analytic help operators capture and analyse network in real time to gain continuous insights, it enables better NM decisions without any delay. It is a key component for enabling real-time automatic closed loops with emerging NFV and SDN.

As a result, our analytic engine is designed to output analytic results immediately and continuously when network data streams are arriving.

2) Data agnostic and self-learning

There are thousands of performance Counters and KPIs for different layers and types of networks, and they have different types of temporal behaviours and data distribution patterns. Additionally, network is a dynamic system, a given KPI behaves differently when topology or other network configuration changes. For example, some KPIs have busy hour patterns following office hours for network nodes close to Industry Parks. Pre data analysis and diagnostics and then selecting or designing specific algorithms for each network KPI is impractical due to large number of KPIs in mobile networks and their dynamic natures.

Existing statistic and ML frameworks, such as R packages contain thousands of algorithms. However, these toolboxes are mainly targeted for data analysts and scientists. For example, the *stats::arima* forecasting function in R has more than ten model configuration and optimization parameters. It is too difficult and complex for software engineers to understand and use it for NM system development without some levels of statistic and ML domain knowledge. The analytic engine should learn and adapt itself to fit input data rather than requiring engineers to manually configure/optimize it for different data sets.

As a result, our design for analytic engine follows the data agnostic and self-learning approach that expects to fit any or most network time series data.

3) Lightweight and self-contained

Many NM systems are large, complex systems and already exist; analytics or anomaly detection feature could be required for many use cases. As a result, the analytic engine should be able to integrate into existing systems without complex changes to existing systems or requiring heavy software/hardware dependencies. In other terms, the analytics engine should be a lightweight component and easy to integrate to systems for different use case scenarios.

A large network could have thousands of network nodes and each node has hundreds of KPIs. For example, for a small mobile network, there would be hundreds of Giga Bytes of raw network data being collected and analysed every day through big data systems. Big data technologies, such as Spark [14], being able to handle streaming data will be normally used. They have some ML capabilities, such as regression and clustering, but do not have in-built anomaly detection algorithms currently. Hence, multiple instances of engine can be executed in parallel by leveraging on existing big data systems to handle large amount of data for these cases.

As a result, our analytic engine is designed as a self-contained library that can be used as a standalone component, and can also be easily integrated into existing big data systems to achieve scalability.

B. *Engine feature and architecture*

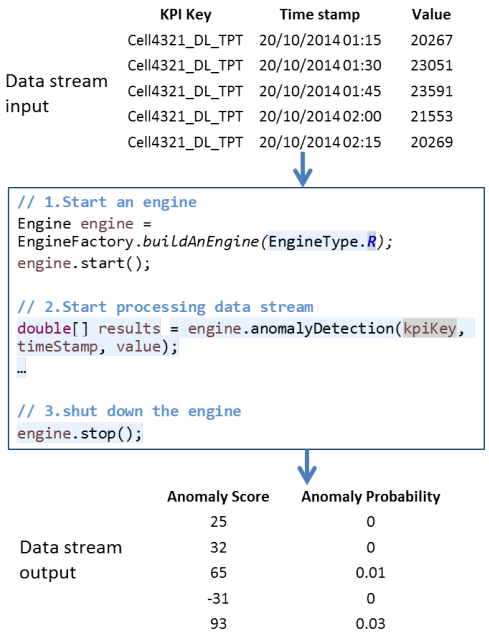

Fig. 1. Simple black box engine API example

Network performance data are monitored and collected in real or near real-time and almost all KPIs are associated with time stamps. As a result, our analytic engine, as input, takes time series data streams; and anomaly detection is one of the key features of the engine. Anomaly detection is about identifying KPI values or observations which do not conform to expected patterns or trends. It could be used in many NM use cases; as examples, a couple of these are discussed in Section IV.

The analytic engine is designed such that it can be integrated as a component in existing NM systems, which are mainly written in mainstream programming languages such Java, C++ (as opposed to languages used in statistic/ML domain such as R). Furthermore, these management systems use Hadoop big data frameworks that are mainly written in Java. Hence the engine is designed with a simple black box Java API interface to facilitate easy integration for engineers with Java competency. As a result, engineers can start using the engine with a few lines of code without any statistic or ML domain knowledge. As shown in Fig.1, the analytics engine takes time series data streams (e.g., KPI) as input, also a unique identification key for each KPI (i.e., a univariate time series) that allows the engine to handle multiple KPIs in parallel. This also helps to determine an assigned engine for a particular KPI when multiple engine instances run in parallel. The engine outputs a numerical array consisting of *Anomaly score* and *Anomaly probability/normalized score* after input every KPI value. Anomaly score can be positive or negative, which represents an anomalous peak and dip respectively, in comparison to normal behaviours. Anomaly probability is a ranged measurement (between 0 to 1) of anomaly which will be described later in the next section. This ranged measurement allows users to filter out low probability or 'trivial' anomalies when it is necessary.

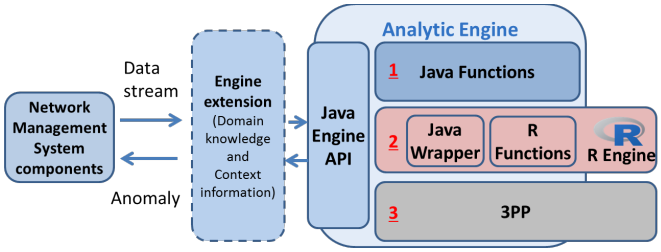

Fig. 2. The analytic engine architecture

As shown in Fig.2, when internal implementations are considered, there are two types of selectable backend engine versions available providing two versions of algorithm implementations; these two versions have two different set of properties. One of them, the R version implemented a complex forecasting based anomaly detection algorithm utilizes certain R packages; however, it requires R engine preinstalled in deployed environment and requires a certain level of computing resources. On the other hand, the Java version implemented a simple heuristic limits based anomaly detection algorithm design but it requires less computing resources. In addition to these two backend implementations, other algorithms or libraries (i.e., 3rd party products) can also be used to extend the engine if needed; this is facilitated by a unified engine Java API interface for different backend engine types as shown in Fig.2. Users or engineers only need to configure one parameter to select/switch between different backend engines. This offers users the flexibility for different requirement scenarios; for example, if R engine is not feasible (or too heavy) for the computing environment, then Java engine type can be selected.

The analytic engine itself is designed as a data agnostic black box. Hence the engine does not require any ML algorithm parameter information or network domain information to process data. However, an optional engine extension can be added to provide more accurate or better results by adding domain knowledge or other contextual information. For example,

- Using an event (e.g., sporting or social event) calendar and cell location information to filter out some anomalous network access peaks during the event, (certain anomalous scenarios such as up/download peaks are expected for cells near sporting venues during sport events).
- On the other hand, for certain KPIs, such as call drop rate anomalous dips do not have any negative service impacts; in such situations anomalies with negative anomaly scores can be ignored.
- High dense anomalies, i.e., a large number of anomalies that happen during a time window possibly indicate "network incidents" that needs to be addressed by taking corrective actions, for example by making changes to the network. Anomalies could be further refined only to keep such high dense anomalies representing "network incidents".

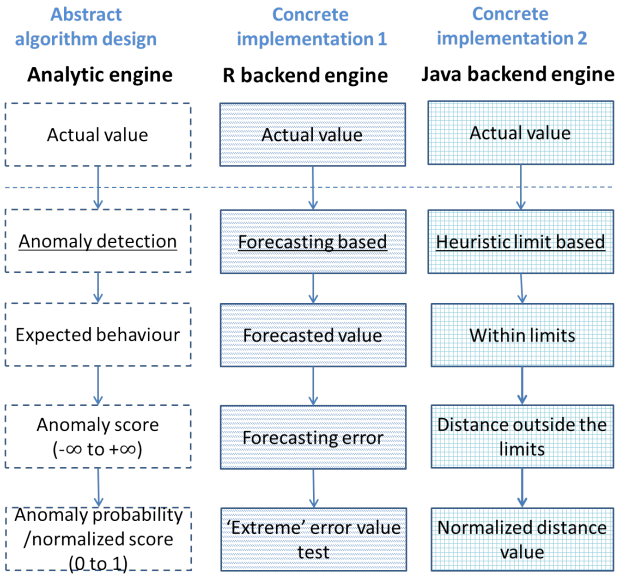

Fig. 3. Abstract algorithm design and concrete implementations

III. Real Time Anomaly Detection Design and Implementation

In this section we describe our innovative algorithm and system design for real time anomaly detection. The algorithm is designed by combining several ML and statistical methods. As described earlier, we have two algorithms implementations for R and Java backend engines. These two implementations follow one abstract anomaly detection algorithm design concept as shown in Fig.3. As a result, two algorithms would take/give same format of input/output with a same analytic engine interface allowing users to select backend implementations depending on their requirement without modifying the system code. Two algorithm implementations will be described in the following subsections.

A. Forecasting based approach

Forecasting based anomaly detection is an algorithm implemented for R backend engine (Fig.3). The implementation utilizes a complex time series forecasting algorithm, i.e. ARIMA, implemented in R *forecast* package.

Forecasting is about making predictions of the future (KPI) values based on past and present data patterns and trends. In forecasting based approach, a one-step ahead forecasted value is used to represent an expected behaviour/value of an incoming actual value (Fig.3). If the forecasted value is "comparable" with the actual value, then we say the actual value is expected or normal, otherwise, we say the value is unexpected or an anomaly. In the following, we firstly describe the ARIMA forecasting algorithm used to get one-step ahead forecasting value for a coming actual value in real time.

Having data agnostic design in mind, ARIMA [15] is used for the forecasting algorithm. This design choice is based on the fact that the ARIMA models are, in theory, the most general class of models for forecasting a time series [15]. Various time series models such as random walk, exponential smoothing and damped trend can be modelled with ARIMA. As a result, we expect more accurate forecasting results when we could have adaptive ARIMA model based module that learns continuously through on-line learning.

Non-seasonal ARMA model consists of two components, an autoregressive (AR) component and a moving average (MA) component. The forecasted value at time t (y_t) is calculated as the sum of the regression of recent values (x), plus the average over previous period random variations or errors (e).

$$y_t = \alpha_1 x_{t-1} + \cdots + \alpha_p x_{t-p} + e_t + \beta_1 e_{t-1} + \cdots + \beta_q e_{t-q}$$

where, α denotes coefficients of the regression model of the p most recent values. β denotes average factors of random variations over q previous periods. Since ARMA models are suitable for time series which are not stationary in mean and variance, ARIMA extends ARMA with an initial differencing procedure ($x'_t = x_t - x_{t-1}$) to reduce the non-stationary. Significant amount of network KPIs have strong seasonal patterns, such as daily/weekly patterns. We use STL method [16] which is robust to anomalies, to remove seasonal components firstly, then applying a non-seasonal ARIMA model for forecasting.

Once we have a forecasted value from ARIMA for an incoming actual value, the difference between the forecasted value and the actual value can be calculated as the anomaly score. It represents a deviation between the expected normal behaviour and actual behaviour. Hence, any of forecasting error measurement could be used for anomaly scores, for example, simple error (*actual – forecasted*).

However, forecasting errors may not sufficient to determine if there are real anomalies in many cases, even the forecasting errors are very large. For example, continuous large forecasting errors could only mean that a forecasting model does not work really well rather than continuous anomalies. As a consequence, a further process step is added to calculate the anomaly probability in the algorithm design to find those single or isolated anomaly scores to give more accurate anomaly detection. The anomaly probability is calculated based on applying statistical test [17] on anomaly scores. We firstly get a t value of an anomaly score s of most recent n number of anomaly scores which are assumed as approximately Gaussian distribution.

$$t = \sqrt{\frac{n(n-2)|(s-mean)/sd|^2}{(n-1)^2 - n|(s-mean)/sd|^2}}$$

where $mean$ and sd are the mean and standard deviation of the recent anomaly scores. Then, we calculate the final anomaly probability ap based a defined significant level s which could represent the sensitivity of anomaly detection (e.g.$s = 0.05$).

$$ap = \begin{cases} \dfrac{s - p_t}{s}, if\ p < s \\ 0, \quad if\ p \geq s \end{cases}$$

where p_t is the p-value for the t-distribution with $n-2$ degrees of the t value. As a result, a single or isolated anomaly score value passed the significant level will indicate a possible anomaly, e.g., anomaly probability is greater than zero. The more "extreme" the anomaly score, the higher the anomaly probability.

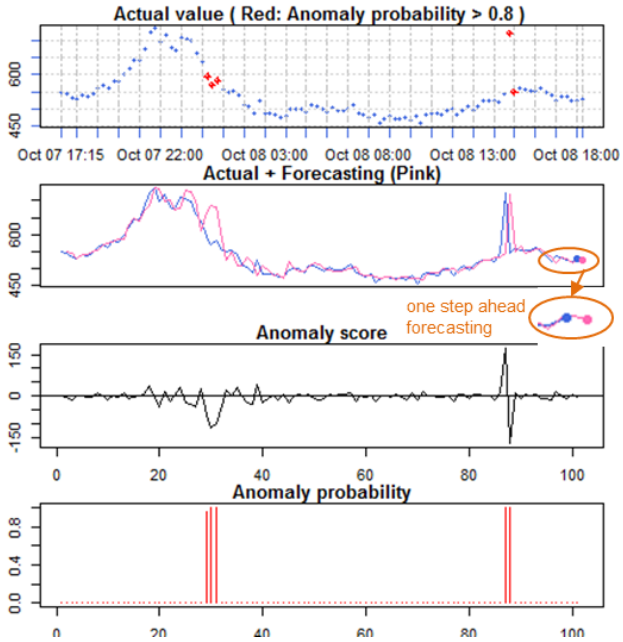

Fig. 4. Forecasting based anomaly detection

By having the above fundamental algorithm ready, we are able to wrap the algorithm to handle streaming data. The idea is to let the engine periodically updates the forecasting model and other parameters by taking most recent stream data into account. Hence, the model is able to adapt itself when new incoming data pattern is changed. When the engine is started, it will create an event counter based model updating trigger to check if the learned model needs to be updated for new incoming data. When the trigger is active, the engine will update a frequency parameter, which is used to identify seasonal patterns that need to be learned. For example, if only few days' data is available, the engine only checks daily

patterns; nonetheless weekly patterns will be checked when more data becomes available. Then the engine will refresh the learned model based on recent data and resets the model update trigger again for a next update phase. Hence, the engine will continuously update the forecasting model by taking most recent data into account to improve forecasting accuracy; improving forecasting accuracy will lead to better anomaly detection accuracy as a result. The simplified engine process flow is depicted in the following pseudo code:

```
Initialize engine cache: kpiValues, forecasted, model, anomalyScores;
do
    value = engine input: a new KPI value;
    // calculate an anomaly score
    if forecasted == null then
        forecasted = value;
    end
    anomalyScore = value − forecasted;
    anomalyScores = append(anomalyScores, anomalyScore);
    // calculate an anomaly probability for the last anomaly score
    anomalyProbability = getAnomalyProbabitly(anomalyScores);
    // calculate a forecasted value for next coming KPI value;
    kpiValues = append(kpiValues, value);
    if modelUpdateTriger() == true then
        frequency = updateFrequency();
        model = arimaModel(kpiValues, frequency);
        resetModelUpdateTriger();
    end
    forecasted = forecast(model, kpiValues, frequency);
    engine output: [anomalyScore, anomalyProbability]
while shutting down the engine;
```

Fig.4 is a system real time output snapshot to illustrate the forecasted based approach. The top plot is the actual KPI time series data with detected anomalous values marked in red. The second plot is a comparison of actual values with forecasted values. The two lower plots show s anomaly scores and anomaly probabilities.

B. Heuristic limit based approach

Heuristic limit based anomaly detection is an algorithm implemented within the Java backend engine as shown in Fig.3. As this algorithm is simple and lightweight, we can easily implement it from scratch in Java without relying on prebuilt statistic libraries, such as R packages to avoid complex dependencies. Since it is lightweight, it consumes much less computational resources in comparison with the forecasting based approach.

The heuristic limits of an incoming actual value are calculated based on historical data and it represents expected normal data boundaries or behaviours. If an actual value is outside of the limits, then it is categorized as an anomaly.

Given a sequence of values X as a time series, the upper and lower limits for finding anomalous values can be calculated based on robust statistics, i.e., median and Median Absolute Deviation (MAD).

$$limit(X) = median(X) \pm 3 * median(|X - median(X)|)$$

To handle seasonal patterns, we assume a constant periodicity p existing in the series. The historical data of current time t, X_t is selected based on the following equation to handle the seasonality for the range functions.

$$X_t = x_{t-f-w}, x_{t-f-w-1}, \dots, x_{t-f}, \dots, x_{t-f+w-1}, x_{t-f+w},$$
$$where\ f = 0, p, 2p, 3p \dots$$

w defines the window size for current time to take account of neighbours' data to increase statistical sample size and also relax the periodicity p value. For example, 7 and 9 o'clock data would be also used for calculating the limits of 8 o'clock time window; a peak normally happening in 7 o'clock occurred in 8 o'clock is still considered as normal. As a result, for example, if there is a weekly pattern, only related time windows of pervious Mondays' data will be used to calculate limits of current Monday's time points.

However, for network KPI values, the distribution of X_t are fairly skewed or asymmetric in most cases based on our observations of many real world networks datasets. Calculating both upper and lower limits using a same formula will lead the limit over estimated for the un-skewed side. For example, a lower limit (the vertical dot line on the left side) is shown in Fig.5. It could cause of missing anomalies for un-skewed side. Hence, we divide original data X_t in two subsets based on the median to find upper and lower limits separately.

$$upper = limit(X'_t)\ , where\ X'_t \subseteq X_t, X'_t \geq median(X_t)$$

$$lower = limit(X'_t)\ , where\ X'_t \subseteq X_t, X'_t \leq median(X_t)$$

The separately calculated upper and lower limits of the distribution are shown in Fig.5. as vertical solid lines. It does not have over estimation problem on the un-skewed side.

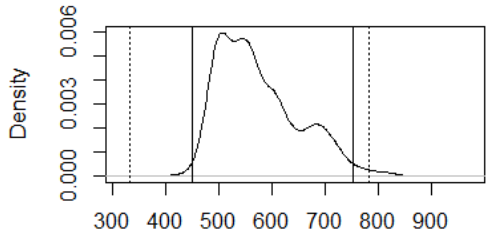

Fig. 5. A KPI value distribution example and calculated limits

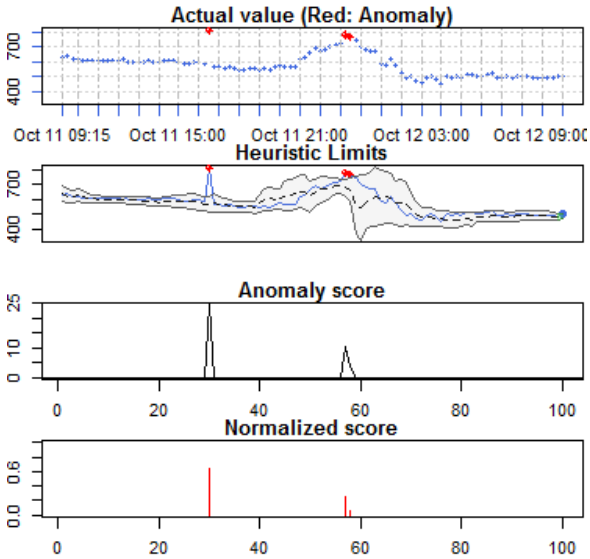

Fig. 6. Heuristic limit based anomaly detection

By having the upper and lower limits for an incoming actual value, a data value outside the limits can be easily detected as anomalies. The distance outside of the limits can be simply calculated as anomaly score $s = \frac{x-upper/lower}{median(|x-median(X)|)}$, which is relative to the MAD.

For heuristic limit based approach, the anomaly probability p is a normalized anomaly score based on recent anomaly scores S to bring the s into a standardized range (0, 1).

$$p(s) = \frac{s - min(S)}{max(S) - min(S)}$$

The Fig.6 illustrates the system output snapshot of heuristic limit based approach. The top plot shows actual KPI time series data with detected anomalies marked in red. The second plot shows heuristic limits calculated in real time for coming values. The two lower plots show anomaly scores and normalized scores.

IV. Use Cases and Evaluation

In this section, we first briefly describe two application scenarios of anomaly detection system in the context of mobile networks. Then we present some evaluation results using our anomaly detection analytic engine.

A. Real-time Network Performance Monitoring

As briefly mentioned earlier, in NM vast amount of different counters exist in different types of nodes with each node having many counters/KPIs/KQIs that represent various performance measurement. These counters, KPIs or KQIs indicate "health" of the node/network and need to be monitored continuously to assure proper functioning of the network. However, given that a large amount of counters exist in networks, anomaly detection systems can continuously monitor all the counters to learn "normal behaviours" for each counters and raise alarms automatically when anomalous values are detected.

As an example, one important KPI is mobile call drop count in a given cell. Mobile call drop could happen due to various reasons such as unexpected radio interference, obstacle for radio signal propagation, frequency spectrum conflicts between neighbouring cells, weather related issues etc. It is important for network engineers to monitor this KPI for all cells and when it is going beyond normal limits for each cell take relevant actions. With existing traditional static threshold based performance monitoring, engineers will only be alerted when the call drop KPI goes to an unacceptable state or a very high value. With our anomaly detection system applied, engineers will be notified when there is an abnormal level of call drop. Hence, engineers or systems can start examine and take actions on network proactively before network going to an unacceptable state.

B. Proactive network resource provisioning

Counters that indicate/measure network load can be analysed by the anomaly detection system to deploy additional network resources when network experiences abnormal peaks. At the same time, when the network load back to normal level, it can trigger shutdown of such additional resources to save energy. These network resources can be, for example, virtual nodes and links (VNFs) deployed in cloud or the resources can be physical nodes such as radio cells, which are placed in overlapping coverage areas to handle additional load in peak periods. One such scenario is shown in Fig.7.

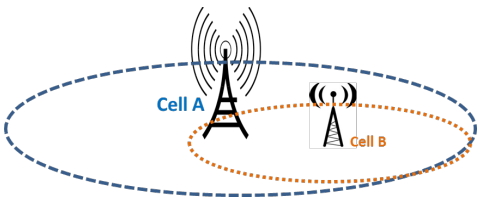

Fig. 7. Cell overlapping coverage

Here cell A is a macro cell covering a large area and cell B is a micro cell covering small area but with dense mobile users (e.g., shopping mall). Cell B is a redundant cell resource and will only be active when cell A is possibly congested or problematic. This can be achieved by first looking at the high density anomalies output that indicates the load at cell A persists at least for a while (and not just short transient problematic conditions) and then based on the anomalies the cell B can be activated or deactivated. As shown in Fig.8, if high density anomalies exist, cell B will be activated as a backup of cell A or to handle excess load from the area. If no high density anomalies exist, the cell B should be deactivated if in an "active" state (in such energy saving scenarios there are other complexities and they are outside the scope of this paper).

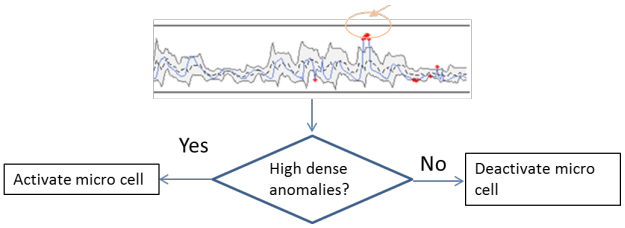

Fig. 8. Micro cell on/off controller based on high dense anomalies

C. Evaluation and discussion

Our implementation is evaluated intensively in prototype testing environment with real network datasets which involve thousands of radio cells and months of data. To avoid privacy and sensitivity issues of real datasets that are not in the public domain, we show and describe results using an open source dataset, and then briefly present the summarized results on large real datasets as well.

A small part of our evaluation is based on an open source network traffic throughput dataset [18]. In this dataset, we assume the network traffic is normal for the entire time of the fourth week (Dec 13-19) and then manually inject some anomalies into the dataset as ground truth to evaluate algorithms; this evaluation strategy is similar to other evaluations such as in [11] [10]. Fig.9 illustrates results of the forecasting based approach. Anomalies injected manually are marked in orange vertical lines in the first plot. The detected anomalies are marked in red dots. The second plot shows a comparison between forecasted and actual values. Alerts can

be sent to other components of NM systems based on anomalies as discussed in the use case scenario.

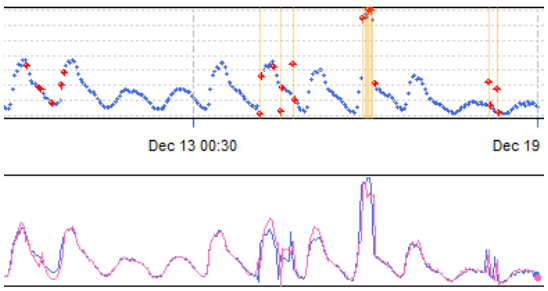

Fig. 9. Fig. Results of forecasting based anomaly detection

Fig.10. illustrates results of the heuristic limit based approach. The second plot shows the calculated upper and lower limits with regard to actual values. Here there is also a scenario that high dense anomalies detected on Dec 16 as 'network incident' indicator which can be used to activate micro-cells as discussed earlier.

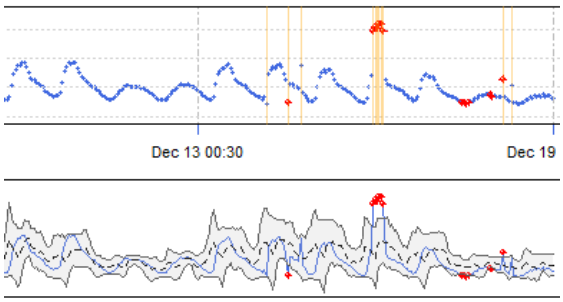

Fig. 10. Results of heuristic limit based anomaly detection

The statistical results are summarized in TABLE I, and show the heuristic limit based approach providing a bit better precision in this case (dataset). From visual inspection of Fig. 9, we could see data points after real anomalies are also detected as anomalies by the forecasting based approach, because of anomalies having strong effects on immediate forecasting results which leads to false positive results in many cases. As a result, false anomaly locations of the forecasting based approach is more close to real anomaly locations in comparison to heuristic limit based approach.

TABLE I. ALGORITHM ACCURACY EVALUATION RESULTS

Algorithm	Precision	Recall
Forecasting based	0.53	0.73
Heuristic limit based	0.57	0.73

With a large real network datasets, evaluated several critical KPIs in both 3G and LTE networks recommended by radio experts. Four basic types of anomalies are injected and labelled in the dataset by a network experts, since time series anomalies can be classified and represented in four basic types (*additive*, *innovative*, *level shift* and *transient change anomaly*) [10]. The final result for all basic types anomalies are shown in TABLE II. Both algorithms give good result in most cases except level shift anomalies. Both algorithms produce low precisions for level shift anomalies. This because of level shift anomalies have permanent subsequent effects; the model takes time to "learn" new "normal" values creating many false positives (low precision) in this intermediate learning phase. However, sometimes, it may make sense to alert subsequent effects for a short period after a level shift. For level shift anomalies, if a short period or a group of points after a level shift location is considered as a single anomaly, then precision would be much higher in this case.

TABLE II. EVALUATION RESULTS OF MOBIEL NETWORK DATASETS

Anomaly basic type	Algorithm	Precision	Recall	F1 score
Additive	Forecasting based	0.69	0.50	0.57
Additive	Heuristic limit based	0.93	0.73	0.78
Innovative	Forecasting based	0.63	0.35	0.45
Innovative	Heuristic limit based	0.76	0.49	0.60
Level shift	Forecasting based	0.08	1.00	0.14
Level shift	Heuristic limit based	0.01	0.67	0.03
Transient	Forecasting based	0.57	0.15	0.24
Transient	Heuristic limit based	0.83	0.41	0.55

V. RELATED WORK

In this section, we classify the related work into two categories and discuss them in relation to our approach. The first category is about generic anomaly detection algorithm implementations or frameworks, and the second category is about mobile network related anomaly detection systems. We discuss them separately in the following.

A. Generic anomaly detection frameworks

Anomaly detection has been studied extensively. There are a large number of algorithm implementations and frameworks exist [19], however, most of them are not targeted or suitable for time series data streams. To the best of our knowledge, our approach is the first analytic engine focused on real-time anomaly detection on time series data streams. In the following, we discuss a few well-known examples.

Rainbow [20] is from authors of the popular time series *forecast* R package. It detects anomaly based on functional depth method, which represents data in curves. Data points are associated with a curve shape and points that do not lie within the range of the majority of curves, are detected as anomalies. However, it is not designed for time series data that has strong seasonal behaviours. *tsoutliers* is a R package focused on time series anomaly detection based on algorithms described in [10]. It designed an iterative algorithm, which starts at fitting an ARIMA, or *stmt* model on an original time series, then find anomalies based on analysis of critical residuals of the fitted model. Then it will fit a new model again on the time series after anomalies are removed. The algorithm will loop through above steps until all anomalies are found. *AnomalyDetection* [11] is an R package developed by Twittes for long-term anomalies owing to a predominant underlying trend component in the time series. It uses a Piecewise Median technique to

estimate a trend from a time series, and then does statistical test on residuals of the time series after the trend and seasonality are removed [11]. However, both *tsoutlier* iterative algorithms and Twittes *AnomalyDetection* for long term anomalies are not designed for real time data streams as most R packages. They cannot detect anomalies for streaming data in real time.

MOA [21] is a well-known open source data stream analytic framework which supports a wide range of algorithms including various anomaly detection algorithms, such as STORM, Abstract-C, COD and MCOD. These algorithms [22] [23] use distance based measure techniques similar to clustering to detect anomalies. It measures density of each object's neighbourhood, objects have no or few neighbours are detected as anomalies. The advantage of these algorithms is that they can be applied to data of arbitrary dimensionality and also in general metric spaces. However, for univariate time series data, which has strong seasonal behaviours, these algorithms are not best candidates, since periodic patterns are more important than neighbour values. Apache SAMOA [24] is a distributed streaming ML framework that contains a programing abstraction for distributed streaming ML algorithms. Hence, it enables development of new ML algorithms without dealing with the complexity of underlying distributed streaming frameworks for ease of scalability. A number of algorithms are developed in SAMOA currently, including AMRules implementation [25] which contains the anomaly detection feature. AMRules algorithm can learn regression rule sets from data streams. However, it is a supervised ML algorithm requiring labelled data which is not available in our use case scenarios.

B. *Anomaly detection systems for mobile network KPIs*

Anomaly detection related to network KPIs is also not a new concept. However, these works [7] [8] [9] translate the problem into supervised ML problems. They follow a two-steps approach; firstly, generating profiles or models based on training datasets for nominal behaviours for all KPIs, and then uses the generated profiles against the testing dataset to determine if there are anomalies. [7] is based on one class support vector machine algorithm. [8] and [9] use ensemble-method approach contains multiple algorithms, such as support vector machine. Our work focuses on real time data streaming and data agonistic, hence, we do not and cannot rely on availability of training datasets for all KPIs.

VI. CONCLUSION

Anomaly detection of streaming data in the context of NM system development has specific feature requirements. Prebuilt anomaly detection libraries such as the ones that are available in R, MLlib do not fulfil these necessary requirements. In this paper we have described design principles based on the requirements and innovative algorithm designs; without such algorithm design that combines a number of statistical functions in a workflow, anomaly detection of time series streaming data cannot be achieved particularly in the context of NM scenarios. The two versions of the algorithms were discussed and we have discussed use-case scenarios showing how our work applies in real world scenarios. Finally using an open data set we have shown the evaluation of the system. We plan other streaming analytic features design and use cases in our future work.

REFERENCES

[1] Tongke, F. Hadoop-Based Data Analysis System Architecture for Telecom Enterprises. in International Conference on Computational and Information Sciences. 2013.

[2] Wang, M. and S.B. Handurukande, RPig: Concise Programming Framework by Integrating R with Pig for Big Data Analytics, in Cloud Computing with E-science Applications, O. Terzo and L. Mossucca, Editors. 2015, CRC Press/Taylor & Francis.

[3] Miyazawa, M. and M. Hayashi. In-network real-time performance monitoring with distributed event processing. in IEEE Network Operations and Management Symposium. 2014.

[4] Jiang, N., et al. Correlating real-time monitoring data for mobile network management. in International Symposium on a World of Wireless, Mobile and Multimedia Networks 2008.

[5] 3GPP TS 32.426 V10.3.0 (2011-03). Technical Specification. 3rd Generation Partnership Project. 2011.

[6] Traynor, P., et al. On Cellular Botnets: Measuring the Impact of Malicious Devices on a Cellular Network Core. in ACM conference on Computer and communications security. 2009.

[7] Zhang, R., et al. Network Anomaly Detection Using One Class Support Vector Machine in International MultiConference of Engineers and Computer Scientists. 2008.

[8] Ciocarlie, G.F., et al. Detecting anomalies in cellular networks using an ensemble method. in International Conference on Network and Service Management. 2013.

[9] Ciocarlie, G., et al. DCAD: Dynamic Cell Anomaly Detection for Operational Cellular Networks. in Network Operations and Management Symposium. 2014.

[10] Chen, C. and L.-M. Liu, *Joint Estimation of Model Parameters and Outlier Effects in Time Series* Journal of the American Statistical Association., 1993. **88**(421): p. 284-297.

[11] Vallis, O., J. Hochenbaum, and A. Kejariwal. A Novel Technique for Long-Term Anomaly Detection in the Cloud. in 6th USENIX Workshop on Hot Topics in Cloud Computing. 2014.

[12] *R project*. Available from: https://www.r-project.org/.

[13] *MLlib*. Available from: http://spark.apache.org/mllib/.

[14] *Apache spark*. Available from: http://spark.apache.org/.

[15] Cao, L. and P.S. Yu, Behavior Computing: Modeling, Analysis, Mining and Decision. 2012: Springer.

[16] Cleveland, R.B., et al., *STL: A Seasonal-Trend Decomposition Procedure Based on Loess.* Journal of Official Statistics, 1990. **6**(1): p. 3–73.

[17] Bedson, P. and T.J.D. Farrant, *Practical Statistics for the Analytical Scientist: A Bench Guide*. 2009: Royal Society of Chemistry.

[18] *Internet traffic data (in bits) from an ISP*. Available from: https://datamarket.com/data/set/232h/#!ds=232h&display=line.

[19] Chandola, V., A. Banerjee, and V. Kumar, *Anomaly Detection: A Survey*. ACM Computing Surveys, 2009. **4**(3): p. Article No. 15.

[20] Hyndman, R.J. and H.L. Shang, *Rainbow plots, bagplots and boxplots for functional data.* Journal of Computational and Graphical Statistics, 2010. **19**(1): p. 29-45.

[21] Bifet, A., et al., *MOA: Massive Online Analysis.* Journal of Machine Learning Research, 2010. **11**: p. 1601-1604.

[22] Georgiadis, D., et al. Continuous Outlier Detection in Data Streams: An Extensible Framework and State-Of-The-Art Algorithms. in ACM SIGMOD International Conference on Management of Data. 2013.

[23] Kontaki, M., et al. Continuous Monitoring of Distance-Based Outliers over Data Streams. in International Conference on Data Engineering. 2011.

[24] Bifet, A. and G.D.F. Morales. Big Data Stream Learning with SAMOA. in IEEE International Conference on Data Mining Workshop. 2014.

[25] Vu, A.T., et al. Distributed Adaptive Model Rules for mining big data streams. in IEEE International Conference on Big Data. 2014.