IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 18, 2021, accepted April 9, 2021, date of publication April 15, 2021, date of current version April 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3073598

Towards the Automatic and Schedule-Aware Alerting of Internetwork Time Series

DANIEL PERDICES[1], **JOSÉ LUIS GARCÍA-DORADO**[1], **JAVIER RAMOS**[1], **RODRIGO DE POOL**[2], **AND JAVIER ARACIL**[1,2]

[1]Department of Electronics and Communications Technology, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 28049 Madrid, Spain
[2]Naudit High Performance Computing and Networking, S. L., 28049 Madrid, Spain

Corresponding author: José Luis García-Dorado (jl.garcia@uam.es)

This work was supported in part by the Ministry of Science and Innovation of Spain through Project AGILEMON under Grant AEI PID2019-104451RB-C21, and in part by the Predoctoral Fellowship of the Program for the Training of University Lecturers of the Ministry of Science, Innovation and Universities of Spain, under Grant FPU19/05678.

ABSTRACT A common factor of every network monitoring system is an alerting module for time series. This module aims at triggering a warning when any type of abnormal behavior is detected in the patterns of a time series. Such a search for anomalies can be carried out by network managers as a supervised task such that the thresholds for considering a measurement as an anomaly are set following a manual process. Alternatively, we focus on how to translate such a task to an unsupervised one, thus alleviating network managers' dedication. To this end, we have developed, based on the experience of monitoring dozens of networks, a player of real anomalies. Thus, by recreating real issues, the alerting systems' parametrization can be carried out without supervision. Additionally, as a novelty, we propose to consider the network managers' workforce as a significant parameter to configure the thresholds of the alerting module—essentially, avoiding triggering alarms that will hardly receive attention. Then, we propose to measure and rank alarms by relevance, and relate them to the time to be solved for constructing, eventually, automatic schedules for the members of the staff—according to their time availability. Finally, all these proposals have been put into practice in various deployments of monitoring systems on networks in operation, which gives us evidence of its usefulness and low demand for resources.

INDEX TERMS Monitoring systems, alerting module, Internetwork time series, LSTM networks, network managers schedule.

I. INTRODUCTION

Modern monitoring systems provide network managers with thousands of time series of diverse measurements on beautiful dashboard environments such as Grafana, Kibana, or Tableau [1]. Such time series may represent internetwork Key Performance Indicators (KPI) such as the bandwidth, round trip times (RTT), TCP-window sizes, or the number of retransmissions [2]. However, it is the duty of network managers to inspect the time series to both assess good performance and find any evidence of anomalies in the network [3]. Unfortunately, network managers' time is limited which calls for systems that help them in such a task. Therefore, monitoring systems provide alerting mechanisms so that network managers only need to inspect time series after an alarm is triggered, so alleviating their burden. Typically, such alarms are triggered if a time series exceeds or drops over/below some given threshold from a baseline [4]. However, from our experience in monitoring large-enterprise networks, we noticed soon that the parametrization of such thresholds engenders a conflict. If thresholds are too narrow, there would be a large number of alarms [5] that exceed network managers' dedication, and they will not be analyzed. In the opposite case, if thresholds are too wide, there would be real anomalies missing.

In this scenario, in our view, the keys to set efficient thresholds are: (i) the probability of a real anomaly not to be detected—i.e., a false negative—is marginal; (ii) the matching of the number of alarms to the analysis capacity of network managers. We emphasize that this last aspect is novel albeit the fact that triggering alarms with no time to be analyzed lacks sense.

To address the first design line—avoiding false negatives—, we noticed that typically setting and updating thresholds used

The associate editor coordinating the review of this manuscript and approving it for publication was Chakchai So-In.

to be a manual process based on previous experience without any automatic support. The research community noticed soon that such an approach, named Fixed Threshold Anomaly Detector [6], was not very useful. For example, a period of high activity, in the case of periodic time series, may not represent an anomaly at all but, simply, the typical busy hour of the network [7]. Even more, given that the task of updating parameters is usually tedious, some thresholds may not be adapted to regular network changes (e.g., more users or different services) over time [8].

From this scenario arose the natural idea of defining dynamics thresholds that the modern anomaly detection systems apply nowadays. The answer to this need for dynamic thresholds has been following a stochastic approach. For example, Prometheus [9] suggests considering time series as Gaussian processes and sets thresholds according to excursions over a certain number of standard deviations over/below the average. The authors in [10] refine that approach by defining the width according to the standard deviation of the noise of the time series—the difference between real and predicted values. Interestingly, the prediction is carried out through a trained Long Short-Term Memory (LSTM) neural network, similar to other works such as [11], [12]. We share their view of how the current trends of transferring machine-learning mechanisms to network problems are a good direction and, then, we embrace the approach of predicting traffic through neural networks.

However, we believe that the way of defining thresholds must be related to real-world anomalies and not to statistical variations of a signal. That is, an anomaly may pass undetected in such approaches, simply, because anomalies do not exceed one or more standard deviations of the time series but, in fact, it may entail a problem for the users of the networks. Fortunately, we have characterized a large number of anomalies in real networks which has allowed us to reproduce the impact of anomalies in regular operation. That is, after replaying anomalies on time series, we can set thresholds in such a way that anomalies are detected. The point of this is that we can define thresholds to maximize/minimize true/false positives ratios of alerting for anomalies. In other words, we are translating a supervised task, where network managers must perform the threshold definition manually, to an unsupervised one, where this is totally automatic and requires minimal validation from them.

In the second line of design—a balanced number of alarms—we note that the alarms must be triggered if an anomaly is detected regardless of whether the number of alarms is excessive. That is, making wider the threshold intervals to reduce the number of alarms is not the point, the key is that alarms can be ranked. Ranked by relevance and dedication time—that time a network manager would require solving it. With this data, dedication and relevance, the set of alarms effectively triggered would be those that are the most relevant for the network management and fit the availability of the management staff. Note that triggering few alarms may render the network managers' workday unproductive, and alarms over the workforce will be, simply, ignored. Needless to say, in the event of a significant number of unattended alarms, the total working capacity should be increased.

To the best of our knowledge, this line of design has received no attention from the Internet community despite its direct relationship with the labor of network managers. As a first approach, we propose to formally define a metric for the relevance of the anomaly. Thus, network managers may order anomalies and take care of the top N anomalies according to their availability. Importantly, note that this avoids defining a hard threshold for what is normal or abnormal behavior of a network which has been an open question since the network monitoring was born [13].

As a further step, we propose to re-adapt such threshold values as a balance between the labor force and the historical time the analysis of an alarm requires. That is, the number of alarms and the time they require must be balanced with the available time of the technical staff. In other words, alarms should be parameterized according to the number and relevance of the resulting anomalies in such a way that managers may pay attention to them. From the network managers' perspective, they simply receive an activity schedule according to relevance and time-dedication functions.

In practice, we have added these two novel design lines in an operational monitoring tool that meets the needs of most of the commercial networks. Throughout the paper, we illustrate the system's operation with examples and evaluate the performance in a real scenario with data from a large network with hundreds of servers for months. In this way, the contributions of the paper can be summarized as:

- **Automatic alerting of internetwork time series**. We explain and illustrate the methodologies and novel mechanisms proposed to set alerting thresholds automatically. Threshold setting is based on the replay of abnormal variations from diverse sets of real incidents and the search for those values that optimize classification metrics. That is, given a trained time series regressor (for example, an LSTM network), we compare its output prediction to the original time series after adding the controlled anomalies. Those thresholds that maximize the identification of the anomalies according to some classification metric (e.g., F-score [14]) are chosen as parameters for the system. This process is completely automated and not based on arbitrary figures.
- **Schedule-aware alerting of internetwork time series**. As a distinguishing characteristic, we have introduced the workforce of the network-manager staff as a relevant element to consider in network monitoring:
  - **Rankings.** We rank anomalies by formally defining their relevance as a function of the deviation of a vector of KPI measurements and the number of triggered alarms. By doing so, it allows network managers to pay attention to the most relevant alarms first.
  - **Schedules.** As a further refinement, we propose a novel schedule-aware alerting approach. We relate

the workforce, relevance, and type of alarm to construct, automatically, schedule proposals for analysis to the network-manager staff.

- **Data and modules availability**. Finally, we share a player of network anomalies together with examples for the reproduction of real-world anomalies fruit of the monitoring of dozens of networks. Besides, a set of anonymized time series used throughout the paper is made available so that the results are reproducible.

In the rest of the paper, we first describe the architecture of the monitoring system and the data under study. Then, we pay attention to the methodologies we have needed to carry out the design lines, and we explain how we have put into practice our proposal. Finally, we study some execution examples and the system's performance in a real scenario, and some future lines of work conclude the paper.

II. MONITORING SYSTEM

As a first step, we describe the architecture and environment of the monitoring system that we have deployed in a diverse set of real-world networks. Such networks include those of banking data centers, oil and gas company infrastructures, and postal and package delivery services. Then, over such deployments, we detail our proposals to promote the automatic and schedule-aware alerting of internetwork time series.

A. ARCHITECTURE

The monitoring system is a probe or set of probes that receive traffic from both SPAN ports of routers and taps that replicate and forward traffic to the capture module on several network interfaces. These interfaces run a tuned high-performance network driver such as HPCAP or Intel's DPDK [15]. These drivers accelerate packet processing workloads running on a wide variety of CPU architectures, especially, multi-core. As a result, they can capture packets and process them at ratios of dozens of Gb/s, even, in commodity hardware.

Once packets are in memory, the monitoring system stores data at three levels [16]:

- **Packet level.** This is the most demanding data set as it requires storing as much volume as the traffic aggregate. Consequently, the system only stores data for a configurable number of days—e.g., one week is the default in our system.
  Alternatively, the system allows capping packets to headers—ruling out payloads—or applying selective capping depending on the application-level relevance. That is, the system stores packet headers in all cases and, additionally, non-encrypted payloads are also stored. Note that encrypted ones do not provide any possible forensic study but still require storage capacity [17]. Packet traces are stored in PCAP format [18] in files that comprise traffic for 5 minutes, after that time, a new file is created. Then, PCAP traces are indexed by name in an InfluxDB database [19].

TABLE 1. Network time-series metrics estimated by default.

Metrics	
Outgoing bandwidth (b/s)	Incoming bandwidth (b/s)
Outgoing bandwidth in packets/s	Incoming bandwidth in packets/s
Number of IP addresses	Number of MAC addresses
Client unanswered packets	Server unanswered packets
Number of Flows	TCP zero windows
Server TCP RST packets	Client TCP RST packets
TCP retransmissions	TCP SYN packets
Duplicate ACK packets	Average RTT per connection (s)

TABLE 2. Examples of records of network metrics of a given IP.

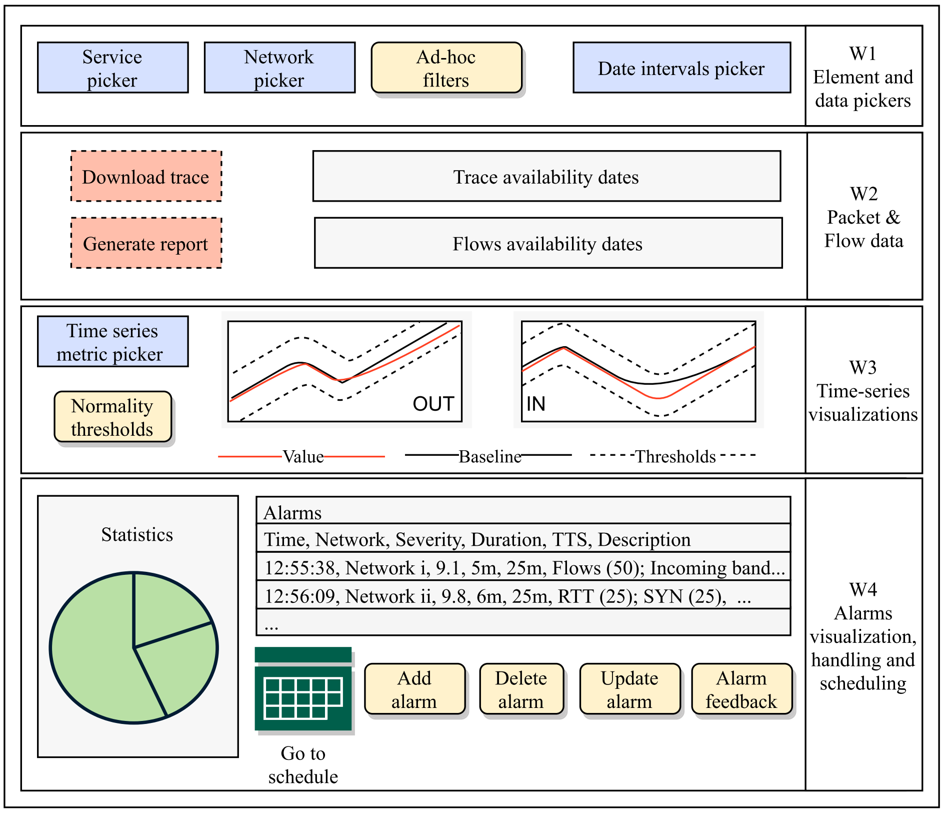
Date (dd-mm-yyyy hh:mm)	Outgoing (b/s)	Incoming (b/s)	... ...	RTT (s)
08-07-2019 19:35	49918	256093		0.0271
08-07-2019 19:40	588324	80718		0.0003
...			...	
25-09-2019 01:50	31479	89977		0.0300

- **Time series.** The data represent a sequence of network measurements over time at different granularities—e.g., 1 second, 5 minutes, 1 week, or 1 month. By default, the time series are constructed by IP address. Under demand, IP addresses can be grouped by ranges or services—hereinafter, a network element. The set of network metrics that the monitoring system measures, by default, are shown in Table 1. Time series' storage needs are far below packet traces. However, we note that storing them at the finest granularity—e.g., 1 second—in networks with a high diversity of IP addresses may be challenging. By default, we store the finest-grained ones for a month, weekly for the last year, and monthly for longer periods. All the time series data is indexed in an Elasticsearch database [20], Table 2 exemplifies this data.
- **Flow-level.** In-between packet traces and time series, network flows summarize traffic by aggregating packets by quintuple (4-layer protocol, IP addresses, and port numbers) so alleviating storage demands [21]. By default, we store traffic measurements as Netflows for one year indexing each of their fields in an Elasticsearch database [22].

B. DATA UNDER ANALYSIS

To illustrate the monitoring system and proposal with numerical examples, we are showing real measurements from the network of an international oil and gas company. The network provides workers, offices, gas stations, and refineries with Internet connectivity and other services such as teleconference, telephony, distribution/sharing files, manage business operations (e.g., Enterprise Resource Planning (ERP) [23]) and customer relations (e.g., SAP [24]), DNS, monitoring protocols, and payment gateways among other services.

In particular, we are analyzing an interval of three months of measurements (March, April, and May, 2020) from more



Service picker
Network picker
Ad-hoc filters
Date intervals picker
W1 Element and data pickers
Download trace
Generate report
Trace availability dates
Flows availability dates
W2 Packet & Flow data
Time series metric picker
Normality thresholds
OUT
IN
Value
Baseline
Thresholds
W3 Time-series visualizations
Statistics
Alarms
Time, Network, Severity, Duration, TTS, Description
12:55:38, Network i, 9.1, 5m, 25m, Flows (50); Incoming band...
12:56:09, Network ii, 9.8, 6m, 25m, RTT (25); SYN (25), ...
...
Go to schedule
Add alarm
Delete alarm
Update alarm
Alarm feedback
W4 Alarms visualization, handling and scheduling

FIGURE 1. Main dashboard of the monitoring system.

than 300 servers and thousands of users. From now on, we will make direct reference to this set for the proposed examples. A sample piece of this data has been anonymized for privacy reasons and is freely available at [25], [26]. We remark that similar deployments are in operation in networks such as international banks, postal and package delivery services, and electric utility companies. Thanks to these deployments, we have both real data of a large number of internetwork time series and, importantly, we have labeled anomalies. That is, while monitoring commercial networks, we have received dozens of notices of abnormal behavior and how they were solved. That has allowed us to create a generator of anomalies following real issues.

C. ACCESS, VISUALIZATION AND ALERTING SYSTEMS

Users can access the different levels of data and inspect them through standard and custom-made modules of Grafana [27]. Grafana is open software to query—with an extensive set of filters—, visualize—at the desired time interval—, alert—based on thresholds—on any kind of time series stored in a diverse set of databases—including InfluxDB, Elasticsearch, MongoDB or SQL-based ones.

This way, Figure 1 provides a conceptual view of the components of our monitoring system's main dashboard. Let us review the four components, or windows, that the dashboard comprises, namely W1... W4.

1) On the left side of the first window (W1), the service, network, or set of networks are introduced as a filter or set of filters based on IP addresses, port numbers or blocks of them—depending on the type of network being monitored. Other more specific filters can be also introduced at this point.
   On the right part, we can choose the date interval for analysis. This operation is the Grafana equivalent of a network filter, it will perform a search over the field that was defined as the timestamp in the database, and only data in the desired interval are considered. In this way, any subsequent query in the databases would be related to the service/network and specific time of interest.
2) Below the first window, we have the Packet and Flow Data Window (W2). As an own module for Grafana, we have created a plugin to download a PCAP trace for the selected networks and dates. As standard Grafana modules do, we have developed a back-end service that, after receiving the search parameters, retrieves the PCAP file or files from the Influx database and concatenates them. Finally, the system serves the filtered trace using HTTP for its download.
   Also in this part, we can download Netflow records for the selected period or, more interesting, creating an automatic report [28]. This functionality extracts from Netflow records an extensive set of KPIs and

depicts them in a formal report. Such indicators include response times of both HTTP and DNS protocols, extended TCP performance metrics, and topology discovery among others.

3) Then, we have a Time-Series Window (W3) for the metrics under measurement. Metrics that can be selected through drop-down menus. By default, the set of available time series are those shown in Table 1, any other measurements that can be extracted from Netflow records or packets can be added on demand.
For each time series, the interface also depicts the baseline and normality thresholds. The former is an estimation for the behavior of the time series and the latter permits to visually examine the limits for normal behavior. We have developed it as a background process that automatically calculates both values, which are inserted in the time series databases as an additional field and it is shown together with the measured values in the time series windows.
4) Finally, the fourth window (W4) is dedicated to the alerting and task scheduling. When a gathered measurement breaks thresholds, an alarm is raised. Each alarm is detailed with its duration, severity/relevance, and estimated time to solve. There is a specific database for alarms deployed in InfluxDB. Such a database permits to view the triggered alarms and some statistics related to the alerting. At the bottom of the figure, the alarm handling interface is depicted. Here a network manager may delete, modify, or mark as solved an alarm as well as, once resolved, add feedback. In addition, at the left of these buttons, it is possible to view the proposed schedule for the network-management staff.

At this point, it emerges the novel goals of this paper: how to determine automatic and better thresholds, how to define alarm relevance, and how to schedule alarms' attention according to the availability of staff of managers. Our proposal to face these aspects is detailed in the following sections.

III. METHODOLOGIES

Let us review both how neural networks may help us to implement precise time series regressors and some concepts about the classification metrics used in our proposal.

A. LONG SHORT-TERM MEMORY NETWORKS

LSTM is an artificial recurrent neural network (RNN) that allows information from previous processing steps to persist and be part of the prediction of new inputs. Thanks to the property of identifying interdependencies in inputs, recurrent networks have been successfully applied in time series prediction [10], [29], [30], speech recognition [31], translation and language modeling [32], areas where context information is necessary to achieve the best results.

Figure 2 illustrates the structure of the recurrent cells of an LSTM network [33]. The vectors X_t, C_t and h_t, respectively, represent the input of the network, its state, and its output at time t.

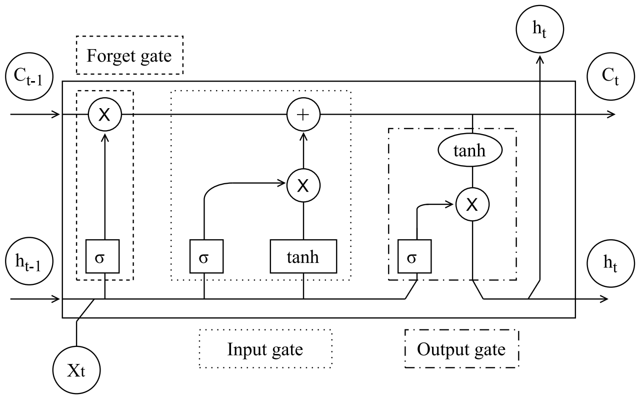


FIGURE 2. LSTM cell diagram. h_{t-1} and C_{t-1} are the cell state propagated from the previous time step. h_t acts both as the hidden state and as the output of the cell.

The C_t state of the network is responsible for storing the context information that will be used in future predictions. For its part, the network is made up of three 'gates':

1) The *forget gate* is responsible for determining which state information will persist in the next state.
2) The *input gate* is responsible for updating the state of the network to include new information regarding the current entry.
3) The *output gate* determines the network output based on the state values (C_t), the input (X_t), and the output of the previous step (h_{t-1}).

Each of the gates of the cell works as a common network *feed-forward*, while the circles in the figure represent the multiplication and sum operations bit by bit. Using gradient descent learning in recurrent networks (*Backpropagation through time*), each gate learns to determine its output to optimize prediction, that is: the *forget gate* will learn what is relevant that persists in the state of the network based on X_t and h_{t-1}; the *input gate* will learn what information should be included in the state based on the same parameters; and, then, the *output gate* will learn how to compose the network output based on the input, the state, and the previous output. Finally, following the cyclical structure of the recurrent networks, the output and its status will be fed into the network along with the next input. LSTM networks are effective time series estimators, even being robust against noisy data, with long-term dependencies and with little relevant or random traits [34].

B. DATA PRE-PROCESSING AND NORMALIZATION

Since neural networks are based on linear regressors, they require that the input data is normalized to provide satisfactory results. Normalization can be carried out in many different ways. The usual standardization is the most common procedure, and it is defined as:

$$X^{(c)} = \frac{X - \mu}{\sigma}, \tag{1}$$

where μ is the mean and σ the standard deviation, so that $X^{(c)}$ has 0 mean and standard deviation 1 as well as no units. Although standard deviation controls the scattering of the data, it does not necessarily bound it—i.e., we can still have high values that bias any linear regressor. On the other hand, min-max scaling is another preprocessing method useful for neural networks,

$$X^{(M)} = \frac{X - \min(X)}{\max(X) - \min(X)}. \tag{2}$$

This method is particularly useful when $\min(X)$ or $\max(X)$ are theoretically bounded—e.g., the minimum measured RTT or bandwidth equal to 0—and it has the advantage of projecting the data to the interval [0, 1] having no units.

C. EVALUATION OF BINARY CLASSIFICATION METRICS

Let us define some metrics widely used in the context of incident classification.

- Sensitivity: Refers to the percentage of true positives to the total positives of the problem. In the specific study case,

$$s = \frac{\text{incidents correctly detected}}{\text{total actual incidents}} \tag{3}$$

- Precision: Refers to the percentage of true positives out of the total positives given by the classifier. In our case,

$$p = \frac{\text{incidents correctly detected}}{\text{incidents detected}} \tag{4}$$

- *F-score*: It is a metric that balances the information of sensitivity and precision. It is calculated as the harmonic mean between the aforementioned sensitivity (3) and the precision (4):

$$F = \frac{s \cdot p}{s + p} \tag{5}$$

  Alternatively, F_β-*score* is the weighted harmonic mean between p and s:

$$F_\beta = \frac{(1 + \beta^2) \cdot s \cdot p}{p \cdot \beta^2 + s} \tag{6}$$

  where the parameter β is a relevance factor that indicates how many times sensitivity is considered more important than precision.

Finally, regarding the evaluation of network anomalies, it is worth noting that incidents tend to occur in bursts. For the purpose of network management in a real-world context, it is far more relevant to be able to detect an incident—i.e., when it begins—than its exact duration. Therefore, when evaluating classifications, we will give an interval as well classified from the first moment in which the incidence is detected and it ends, without considering if during such periods some time intervals are considered non-incident.

IV. PROPOSAL AND OPERATION

In general terms, the operation of an alerting system is as follows: A time series regressor, trained with previous values, predicts the values that would be expected under 'typical' circumstances for the time-series under study. If the system observes that the expected and measured values deviate excessively, then it reports that some issue has occurred. The problem here is to formally define 'excessively'.

Moreover, some incidents may moderately affect more than one time series instead of affecting intensely just one. Actually, a significant hint of a real anomaly is the change in several metrics at the same time [16]. For example, an increase in bandwidth may mean, at a certain time, that there are more users in the network—for example, when workers arrive at offices, which is not an anomaly. However, if at the same time, the number of IP/MAC addresses does not increase, that points to abnormal behavior. This multi-metric approach will demand a process of normalization of the measurements as the values of the metrics are in different units—e.g., bandwidth can be in Mb/s or Gb/s and the number of MAC addresses is in single units.

Moreover, the capacity of handling alerts must be a key factor for an alerting module. In particular, we propose to rank alarms according to their relevance, hence network managers may prioritize. And, finally, if possible, to construct a schedule for them once the time to resolve alarms can be estimated through alarms' characteristics. Let us detail how our proposal copes with these issues.

A. MULTI-METRIC ANALYSIS OF TIME SERIES

Intending to capture time-series covariances, we propose not to contrast time series individually. Instead, we propose to construct a vector of prediction values—through regression—, and, then, to contrast it to the measured vector—i.e., the measured values for the set of time series joined as a multidimensional sample. Let us refer to this latter vector as the vector of network metrics, and any of its single components, a network metric (e.g., 16 components as in our case, Table 1). More formally, we will denote the time series that defines the network metrics by $\{X_k\}_{k=1}^{\infty}$, where $X_k \in \mathbb{R}^d$ is the vector of network metrics in the k-th time interval with d different types of measurements. First, we need to estimate the next X_i based on previous information, this means that

$$\hat{X}_i = \mathbb{E}\left[X_i | X_{i-1}, \ldots, X_{i-m}\right], \tag{7}$$

where m is the number of previous elements we consider to compute the estimate. The next subsection will cover how to compute it, but we want to state our method in a way that is completely agnostic to the method for estimation as long as it is accurate enough. Because the regressor is primarily trained with incidence-free data, the estimated network vector $\hat{X}_i$ will resemble server behavior under normal circumstances. Thus, we define an incidence as a deviation from this expected behavior.

To calculate such deviation between X_i and $\hat{X}_i$, we use the mean squared error (MSE) and compare it to a threshold T,

$$\mathrm{MSE}(X_i, \hat{X}_i) = \frac{1}{n}\sum_{j=1}^{d}(\hat{X}_i^j - X_i^j)^2 > T^2. \tag{8}$$

If the inequality (8) is satisfied, we will state that the behavior of the network in the i-th time interval is suspected of incidence. Consequently, the system would trigger a process to create an alarm in the monitoring system.

Regarding the units of T, we will assume that X and $\hat{X}$ have been pre-processed so that this T has no units. Depending on the normalization procedure, we have different properties. For the standardization (1), the distance is the mean number of standard deviations that separates X_i from $\hat{X}_i$. For instance, a difference of $T = 25$ would mean that all metrics are separated from the expected vector in an average of 5 standard deviations. On the other hand, the min-max scaler (2) uses the amplitude ($\max(X) - \min(X)$) as scale—i.e., a distance of $T = 0.8$ would mean that, in average, metrics are separated 0.8 times the amplitude. The use of one pre-processing technique over others depends on the nature of the metric, since both depend on the accurate estimation of either the standard deviation or the maximum and minimum of the metric.

B. REGRESSOR IMPLEMENTATION

There are many alternatives for time series forecasting, they range from simple methodologies based on moving-average processes [35], through the analysis of complex networks [36], to the newest deep learning-based regressors as previously stated. As mentioned before, we intend to be as agnostic as possible in this regard, therefore our proposal supports any kind of time series prediction technique.

In general, all forecasting techniques want to find a function g such that:

$$g = \arg\min_f \mathbb{E}\left[(X_i - f(X_{i-1}, \ldots, X_{i-m}))^2\right], \tag{9}$$

i.e., g must predict X_i only using the previous m time steps and it must minimize this expectation, which is just the poblational equivalent of the MSE. This means that $g(X_{t-1}, \ldots, X_{t-m})$ provides an estimation of the expectation of the right-hand side of (7),

$$\mathbb{E}\left[X_t | X_{t-1} = x_{t-1}, \ldots, X_{t-m} = x_{t-m}\right] \approx g(x_{t-1}, \ldots, x_{t-m}) \tag{10}$$

For the sake of simplicity, we have entrusted LSTM regressors with the task of predicting time series. They are built with Keras Deep Learning library in Python [37], [38]. The data under study is detailed in Section II-B and the best parametrization per metric and network was obtained using a grid search.

C. THRESHOLD DEFINITION STRATEGY

The threshold T is a parameter that states a trade-off between sensitivity and precision. A high threshold will improve model precision at the expense of its sensitivity by reporting only the most extreme alarms. Conversely, a low threshold will increase the model's sensitivity, decreasing its precision by detecting the finest incidents.

We believe that the alerting thresholds for anomaly deviations must be the result of maximizing classification precision scores of labeled anomalies. Unfortunately, we do not believe it is a good practice to charge managers with the responsibility for labeling incidents. Actually, we believe that the most convenient approach is the opposite. That is, we propose an unsupervised approach that helps managers in their tasks with no more additional burden. To do so, we are adding synthetic anomalies based on our expertise over the regular traffic—picking time intervals of normal operation—and estimate the threshold values that maximize classification scores. In this way, we can provide automatic thresholds regardless of the particular shape or behavior of a network time series. Moreover, we based thresholds on real anomalies, not in statistical deviations that potentially may reflect a typical behavior for heterogeneous time series. Consequently, we have developed an anomaly generator—or, interchangeably, anomaly player—that is parametrized according to real issues observed for years in a large number of networks. It is freely available at [39]. Let us now detail it more formally.

1) GENERATING INCIDENTS

We will assume that we can find—actually, select—some time spans free of incidents. Although it may seem like a strong hypothesis, commercial and industrial networks tend to work most of the time with no incidents and if not, network managers would receive notifications through ticketing systems.

Based on this, we design anomalies according to four parameters:

- Network metric involved.
- The time interval in which the incident occurs.
- The percentage of elements affected—i.e., IP addresses, port numbers or services (set of IP addresses) affected by the incident.
- A multiplicative average factor to apply on the values of the time series. This factor indicates the intensity of the incidence in the affected time series.

Importantly, the induced anomalies can overlap each other, thus affecting several network metrics at the same time. By combining different lists of parameters—fixed according to previous real incidents—we can replay real issues under demand. For example, to simulate a denial-of-service attack, it would suffice to increase the number of connections and incoming bandwidth in a specific interval and set of IP addresses.

In practical terms, our generator receives a JSON file for the description of one or several anomalies. The file includes a set of blocks where each of them comprises four attribute-value pairs describing how the generator has

TABLE 3. Specifications of the set of incident packages.

Incident Package	From (hh:mm:ss)	Until (hh:mm:ss)	Metric	Elements affected	Intensity factor
#1	00:00:00	03:00:00	Incoming bandwidth	80%	×3
	04:00:00	23:00:00	Incoming bandwidth	90%	×2
#2	00:00:00	03:00:00	RTT	90%	×3
	12:00:00	16:00:00	RTT	60%	×4
#3	04:00:00	12:00:00	Retransmissions	100%	×1.5
	08:00:00 + 1 day	12:00:00 + 1 day	Retransmissions	90%	×2
	03:00:00 + 2 days	23:00:00 + 2 day	Retransmissions	70%	×3
#4	00:00:00	03:00:00	Flows	100%	×3
	12:00:00	23:00:00	Flows	100%	×10
#5	06:30:00	15:30:00	Flows	100%	×4
	07:00:00	16:00:00	RTT	90%	×4
	07:00:00	16:00:00	Retransmissions	70%	×6
	07:00:00	15:00:00	Incoming bandwidth	80%	×3

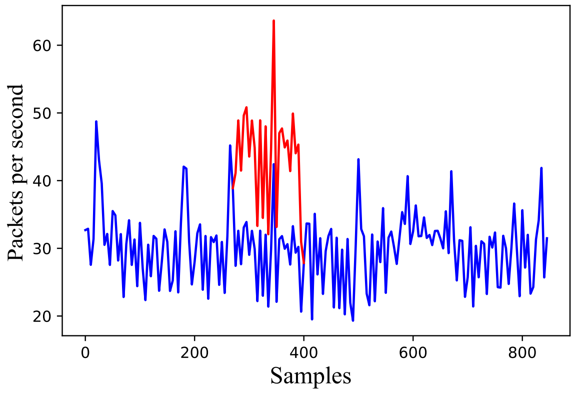


FIGURE 3. Incidence induced the incoming packets-per-second time series.

to modify the network time series. Interestingly, with this approach, we have translated a problem that required supervision into an unsupervised one. On the one hand, any record that has not been altered is labeled as incident-free. Whereas those in which a variation was induced are classified as anomalies. The classification that this generator induces allows evaluating the performance of the alerting system according to how thresholds are parametrized. As an example of the incident generator, Figure 3 depicts an incidence generated on real data. Over the original data, we have an interval of 100 seconds in which the incidence was induced, in particular, an increase in the rate of incoming packets per second. In this example, we illustrate the incidence of a single metric, however, as stated before, incidents can alter several metrics simultaneously.

2) INCIDENT PACKAGES' PARAMETRIZATION

We have translated many of the issues that we have reported in the process of monitoring commercial networks to JSON anomaly-description files. In this way, real issues can be replayed in a controlled environment. Let us refer to them such as Incident Packages.

To illustrate some of the set of incidents we have constructed, the available code includes five examples of Incident Packages [39]. Such examples consist of significant incidents we found in the networks on monitoring. Leveraging on such a set, we provide the evaluation of our proposal in the next section.

While Table 3 summarizes such examples, let us provide more details:

- The Incident Package 1 represents an incident whereby several SAP servers started to carry out data replications unnecessarily.
- The Incident Package 2 arises from a satellite link that began to exceed its regular values of propagation delay.
- For its part, the Incident Package 3 was the result of the saturation of a VoIP gateway.
- Then, in the Incident Package 4, a payment gateway collapsed. Multiple errors on charging clients occurred which generated numerous re-connections, and, eventually, the number of concurrent flows peaked.
- Finally, the Incident Package 5 emerged from visualization problems of multimedia services at end clients. The problem was a saturated link and, consequently, several network metrics varied.

3) SETTING THRESHOLDS

We apply the following process to finally set thresholds per element, i.e., IP address, set of them, or service:

- We feed the LSTM network-based regressor to obtain the estimated time series.
- We apply an extensive set of incident packages to the full set of measured time series. Additionally, to add some extra variety, we randomly vary the number of elements affected—IP address, port number or services as stated before—, intensity factors and times, and reapply them to the time series.
- We contrast estimated and modified time series values. For all possible threshold values in the range, e.g., $[1, 50]$—in terms of standard deviations—we calculate

the F_2*-score*. That is, if given a certain threshold, those periods of time that we know that are anomalous are detected as so—i.e., being over/below the threshold.

- Finally, we select the specific threshold with the best F_2*-score* and analyze its suitability—therefore, skewing relevance towards the sensitivity.

As an example of the output of the above process, Table 4 shows the resulting thresholds and classification metrics per Incident Package for a representative server under monitoring of the available data set (Section II-B). In particular, the server with the highest volume of traffic generated (Top 1). By default, we use a random proportion of 70% of the server's data for training the LSTM models and the remaining 30% as the validation set—to set the thresholds injecting anomalies. The table depicts the best thresholds according to the F_2-score, while precision and sensitivity measurements are also shown. We remark that, especially, the two first metrics reach significant figures. Moreover, we note that in the example the thresholds for the element under study are estimated per Incident Package—or, in other words, per type of incident.

By the time the monitoring system is in real operation, a single threshold per element must be set. Typically, the lowest figure is a good approach so that minimizing false negatives. This will lead to more alarms per item, but note that, thanks to our approach, we can rank alarms and distribute only the most relevant according to the staff's capacity. The real problem would be the opposite, i.e., real anomalies missed. Alternatively, thresholds may vary over time on an intra-day basis, being stricter when more analysis capacity is available, and relaxing them otherwise.

Finally, for the sake of completeness, Table 4 also includes the results for the 10 and 100 most active servers, as well as for the full set of 300 servers under study. In particular, the averages of the classification metrics are shown. Note that, in operation, each server would have its own threshold depending on the optimization process. It becomes apparent that the resulting metrics reach significant figures in these scenarios.

TABLE 4. Examples of optimal threshold selection and classification metrics.

Server	Incident Package	Threshold	F_2-score	Sensitivity (%)	Precision (%)
Top 1	#1	2.55	0.873	88.37	83.21
	#2	5.03	0.953	94.69	97.64
	#3	3.33	0.933	99.58	74.45
	#4	5.01	0.930	99.24	74.43
	#5	2.57	0.848	88.55	72.41
Top 10	Avg.	-	0.897	92.01	81.34
Top 100	Avg.	-	0.904	92.66	82.23
300	Avg.	-	0.889	90.17	84.21

D. ALARMS RANKING

A data center or a large deployment can be made up of thousands of servers or networks to be monitored. Therefore, prioritizing incidents allows managers to focus on those alarms that are more urgent or show more unusual behavior. This calls for a mechanism to compare alarms from different networked elements that were triggered with different thresholds—those that optimized F_2-score for each element in the period under study as the previous section explained.

To make alarms from different elements and times comparable, we propose to relativize the relevance of an anomaly to the particular threshold used by the given element to consider a measurement as an anomaly. This way, we consider deviations in terms of how many thresholds instead of deviations in absolute terms. We define the relevance at time i, R_i, of an alarm observed for a given element as

$$R_i = \frac{\mathrm{MSE}(\hat{X}_i, X_i)}{T^2}, \tag{11}$$

where the metric R_i defines how many thresholds (T) the observed values (X_i) has deviated from the expected ones ($\hat{X}_i$) for the n metrics under consideration. Note that T value is individually computed for each element.

Finally, we consider, R, as the relevance of an observed alarm for a given element during its whole lifetime—as an alarm may be active during several consecutive time intervals $t = 1, \ldots, I$:

$$R = \max_{i=1,\ldots,I} R_i, \tag{12}$$

Clearly, the greater the factor R is, the more radical the behavior of the incidence comparatively is, and then the alarm is more relevant.

Additionally, we are interested in a measure to summarize the relevance of each metric in a given anomaly. We believe that such a measure can be a good descriptor for types of anomalies. For this, it is enough to consider the percentage that each component of the vector $(\hat{X}_i - X_i)$ contributes to its module. The contribution of the j-th metric would be:

$$M_i^j = \frac{(\hat{X}_i^j - X_i^j)^2}{\sum_{j=1}^{d}(\hat{X}_i^j - X_i^j)^2}. \tag{13}$$

The factor M_i^j allows us to order the network metrics according to how much they have deviated from their expected value. Then, we have that those metrics with a higher factor M will have more unusual behaviors and, probably, they will be more descriptive regarding the nature of the incidence. By doing so, the model will not only allow us to indicate if a network exhibits abnormal behavior but, both ordering the anomalies by relevance and estimating the weight of each component.

The first measure, R, will be useful to prioritize one incident over others. The second one, M, will be useful to describe an incident and eventually relate such a description to the time to solve incidents. As a conclusion, we propose using both metrics to plan network managers' schedules.

E. SCHEDULING MODULE

After an alarm is triggered, the monitoring system gets its relevance (R) and the contribution of each metric (M). As time passes, we will also have information about the time each alarm needs to be solved and some statistics of how many alarms are triggered per element. These pieces of information are useful to construct schedules for network managers.

In particular, we note that the frequency that the pair element and metric generates an alarm may be a natural indicator of a heterogeneous behavior. In this way, we propose to give extra relevance to those elements that triggered fewer alarms than those that frequently are requiring inspection. Consequently, the metric R is weighted, according to the number of alarms per element and period of measurement, namely $\bar{R}$. While there are multiple weight functions (e.g., [40], [41]), in particular, we define the weighted metric as:

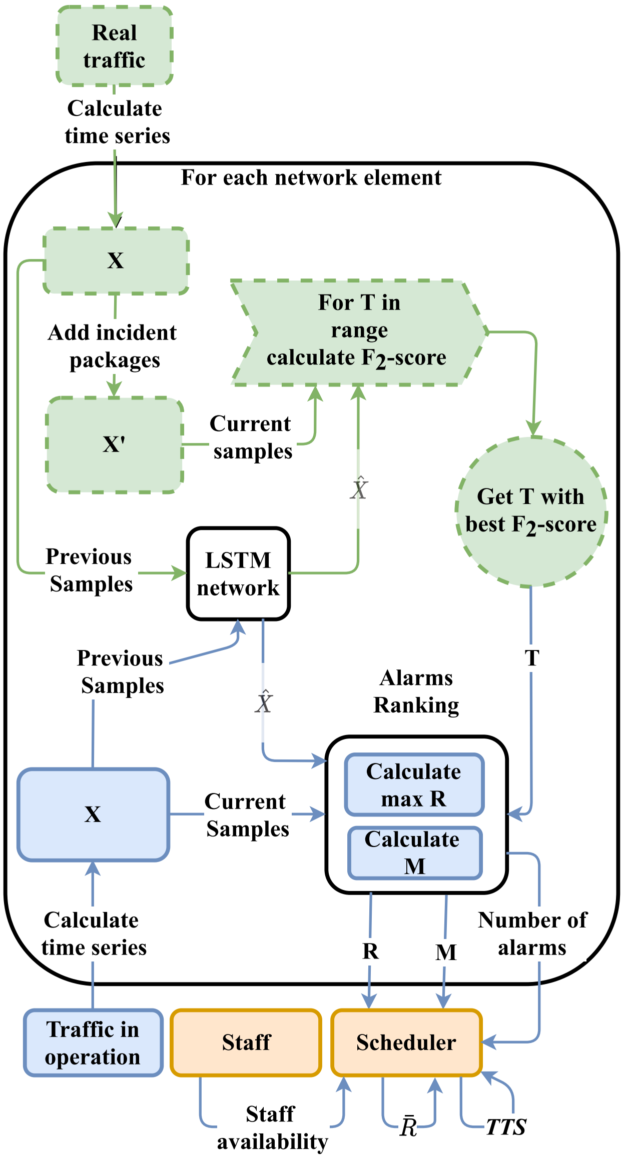
$$\bar{R} = R \log_2 \left(\frac{N}{N_s} \right), \tag{14}$$

where N and N_s stand for the total number of alarms and the number of alarms raised by element or system s. We note that this normalization keeps the original relevance if the percentage of total alarms raised by s is low and reduces the relevance when elements are raising a large number of alarms.

Regarding M, it has so many dimensions as network metrics, now let us consider the value of each dimension as a point in a hyperplane. Then, we propose to relate times to solve (hereinafter, *TTS*) to such points by applying a multivariate linear regression [42], which will give the estimation we pursue. In other words, metric M provides us with a description of the types of network issues, and then, we relate it to *TTS*—as more and more issues are solved. Such a relationship will allow regressing dedication times for present alarms. Finally, to associate alarms under inspection to a network manager, the simplest way is to rank alarms according to both $\bar{R}$ and *TTS*, and managers should go picking alarms sequentially during their workday.

Alternatively, the monitoring system has a module to automatically schedule their tasks. This module considers, in addition to $\bar{R}$ and *TTS*, the availability of the staff. Once staff availability is introduced to the system, the scheduling module will provide a proposal that minimizes the completion time of the most relevant incidents. The module also allows manual modifications by network managers when unexpected issues occur. Similarly, managers can update schedules when they finish a task earlier or later than expected.

To conclude and provide a global view of our proposal, all the previously described processes and parameters of our proposed system are depicted in Figure 4 together with a table that explains the notation. As a summary, in a first stage, alarm thresholds for each network element are calculated using real traffic modified with a wide set of tailored incident packages—shown as dashed boxes in the figure. Next, such thresholds are used in normal operation to trigger and rank alarms—shown as continuous boxes. Finally, the most relevant alarms are scheduled and assigned to managers to be addressed depending on the number of alarms, relevance, time to solve, and workforce (i.e., the set of network managers and their availability)—shown at the bottom of the figure.



Notation	Description
X	Measured time series
$\hat{X}$	Predicted time series
X′	Measured time series with anomalies
T	Threshold
R	Alarm relevance
M	Alarm description
$\bar{R}$	Weighted relevance
TTS	Time to solve

FIGURE 4. Workflow diagram of the proposed scheduling system. Dashed boxes represent the threshold-setting process and continuous boxes represent the system's operation.

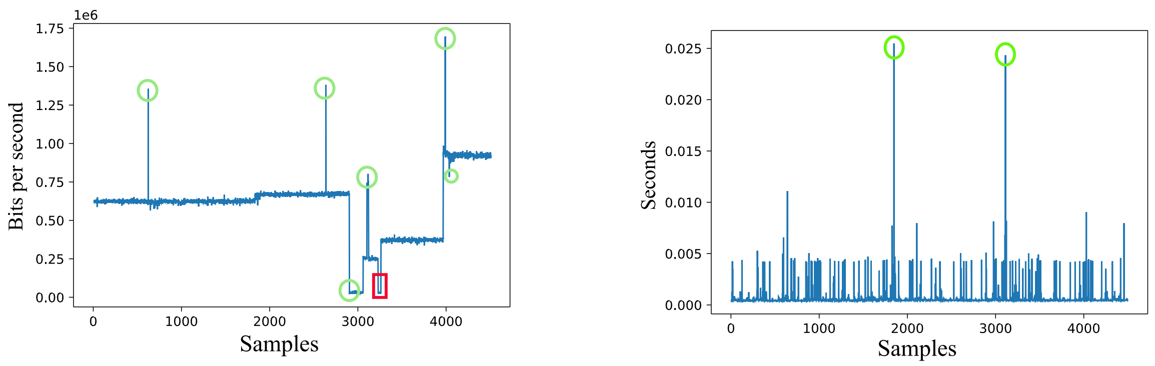
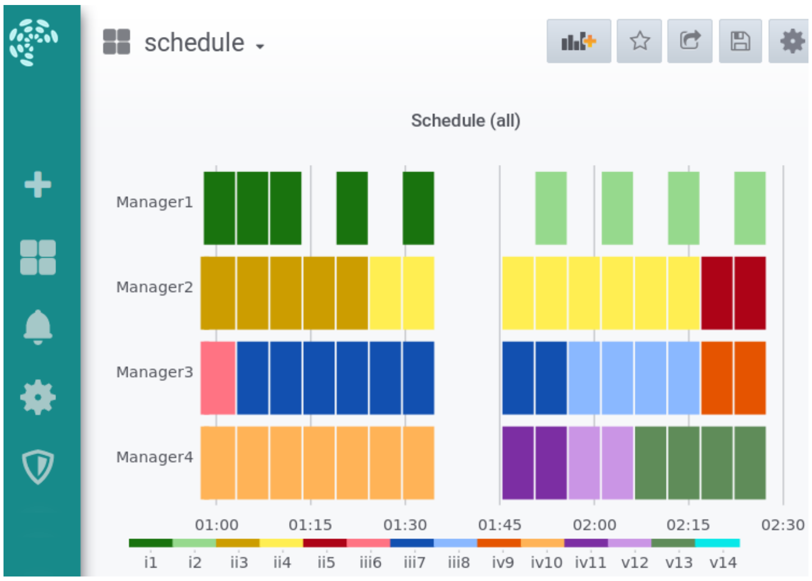


FIGURE 5. Example of anomalies in a given server (outgoing bandwidth and average RTT time series, respectively).



Incident Num.	Element	$\bar{R}$	TTS (min.)
1	i	8.1	25
2	i	4.1	20
3	ii	9.8	25
4	ii	8.1	40
5	ii	2.9	40
6	iii	9.4	5
7	iii	8.5	40
8	iii	5.5	20
9	iv	0.8	40
10	iv	9.0	35
11	iv	7.5	10
12	v	6.9	10
13	v	3.3	50
14	v	0.4	10

FIGURE 6. Example of the schedule module output and inputs.

V. MONITOR SYSTEM IN OPERATION

This section exemplifies the system operation and evaluates its computational load in a real context. In particular, we show how the system has worked for measurements of the first two weeks of June, 2020 once parametrized as Section IV-C3 detailed with three months worth of data, March-May, 2020 (Section II-B).

A. EXECUTION EXAMPLE

In this part, we visually inspect the alarms the system triggered and show how the system provides schedules to the network managers. In the set of 300 servers under measurement, we found an average of 10 incidents per server for the 14 days under evaluation. The server with the fewest incidents had only 3 incidents, and the server with the highest number reported 29 anomalies.

Through manual inspection, we assess that all detected alarms match abnormal behaviors. As an example, we depict the server with the highest number of incidents in Figure 5, where the incidents detected in one of the days under inspection are marked. In circles, we mark the incidents detected by the model, which clearly constitute circumstances worthy of attention by a manager. With a square, we highlight an example of apparently abnormal behavior that was not spotted by the system as such. Digging into this, we noticed that very short drops in the profile of the metric were not infrequent, and they should not be especially treated.

Finally, to illustrate the output that network-management staff would receive, Figure 6 depicts how the scheduling module of the monitoring system provides a proposal. In particular, it has been developed as a custom-made plugin for Grafana (Section II-C). Given the number of elements and days under study, we show, as an example, the system's output for November 1st, 2020 between 1 p.m. and 2:30 p.m. In such an interval, four managers (Manager1 ... Manager4) were active with full availability except for Manager1 with some free intervals as well as a shared established 10-minute break time. At that moment, 14 alarms (i1 ... v14) were active on five network elements ($i \ldots v$). Next to the figure, the inputs that the system uses to create schedules are shown (*TTS* and $\bar{R}$ (Eq. 14)).

B. COMPUTATION TIME

The number of elements that a data center may comprise can range from a few hundred to hundreds of thousands of servers, links, or other devices [43]. Therefore, we have estimated the time to both modeling and applying the model per element which, consequently, allows the extrapolation to different scenarios. In the same vein of generalizing results, all the benchmark was carried out on a single desktop PC, thus encompassing a worst-case hardware scenario.

The modeling using LSTM networks required, on average, less than 30 seconds to model a time series. Then, the evaluation of the possible thresholds took less than 1 second per time series assuming a range from 0 to 50 standard deviations with a step of 0.01. As the modeling of the full set of servers and time series can be an offline process that may require a weekly or monthly update, this should not be a challenge. Once the model is constructed, the evaluation of a potential anomaly of each multi-metric sample requires less than 200 μs on average with marginal variance. Given that the records are typically collected every minute [44], or even every 5 minutes [45], we conclude that the monitoring system may be integrated into the most demanding data centers or networks.

VI. CONCLUSION

Throughout this paper we have presented some novel design lines for the development and deployment of modern monitoring systems in large networks. The design key that conducts all this proposal is to make network managers' work easier by alleviating the dedication that some of their tasks usually require.

In particular, over an already-in-operation monitoring system, we have explained the set of methodologies and mechanisms used to implement novel features such as the automatic and schedule-aware setting of network thresholds for alerting internetwork time series. The full monitoring system has been in operation for months providing us with illustrative numerical examples, evidence of its usefulness, and low demand for resources. Consequently, we believe that our approach may fit in the most demanding scenarios.

As future work, we plan to work on mechanisms that allow network managers to provide richer feedback in comparison with the couple of numbers—the alarm relevance and the time required to solve—that they currently provide as a report once an incident is closed. Such feedback must be close to natural language for the convenience of managers, and, ideally, it should be automatically and rapidly integrated to refine the parameters of the system on the fly. This may be the key to provide the most useful and realistic working schedules.

REFERENCES

[1] S. K. Peddoju and H. Upadhyay, "Evaluation of IoT data visualizationtools and techniques," in *Data Visualization: Trends and Challenges Toward Multidisciplinary Perception*. Singapore: Springer, 2020.

[2] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Experiences of Internet traffic monitoring with Tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, May 2011.

[3] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proc. 1st ACM SIGCOMM Workshop Internet Meas. (IMW)*, 2001, pp. 69–73.

[4] R. Mijumbi, A. Asthana, M. Koivunen, F. Haiyong, and Z. Norman, "DARN: Dynamic baselines for real-time network monitoring," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Jun. 2018, pp. 37–45.

[5] J. Viinikka, H. Debar, L. Mé, A. Lehikoinen, and M. Tarvainen, "Processing intrusion detection alert aggregates with time series modeling," *Inf. Fusion*, vol. 10, no. 4, pp. 312–324, Oct. 2009.

[6] M. Mobilio, M. Orrù, O. Riganelli, A. Tundo, and L. Mariani, "Anomaly detection as-a-service," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2019, pp. 193–199.

[7] F. Mata, P. Žuraniewski, M. Mandjes, and M. Mellia, "Anomaly detection in diurnal data," *Comput. Netw.*, vol. 60, pp. 187–200, Feb. 2014.

[8] J. L. García-Dorado, J. A. Hernandez, J. Aracil, J. E. López de Vergara, F. J. Monserrat, E. Robles, and T. P. de Miguel, "On the duration and spatial characteristics of Internet traffic measurement experiments," *IEEE Commun. Mag.*, vol. 46, no. 11, pp. 148–155, Nov. 2008.

[9] Prometheus. (2020). *How to Use Prometheus for Anomaly Detection in GitLab*. [Online]. Available: https://about.gitlab.com/blog/2019/07/23/anomaly-detection-using-prometheus/

[10] R. Mijumbi, A. Asthana, M. Koivunen, F. Haiyong, and Q. Zhu, "Design, implementation, and evaluation of learning algorithms for dynamic real-time network monitoring," *Int. J. Netw. Manage.*, p. e2108, Mar. 2020.

[11] G. Nguyen, S. Dlugolinsky, V. Tran, and A. López García, "Deep learning for proactive network monitoring and security protection," *IEEE Access*, vol. 8, pp. 19696–19716, 2020.

[12] C.-T. Yang, J.-C. Liu, E. Kristiani, M.-L. Liu, I. You, and G. Pau, "NetFlow monitoring and cyberattack detection using deep learning with Ceph," *IEEE Access*, vol. 8, pp. 7842–7850, 2020.

[13] J. L. García-Dorado, "Bandwidth measurements within the cloud: Characterizing regular behaviors and correlating downtimes," *ACM Trans. Internet Technol.*, vol. 17, no. 4, pp. 1–25, Sep. 2017.

[14] J. Lever, M. Krzywinski, and N. Altman, "Classification evaluation," *Nature Methods*, vol. 13, no. 8, pp. 603–604, 2016.

[15] V. Moreno, J. Ramos, P. M. Santiago del Río, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Commodity packet capture engines: Tutorial, cookbook and applicability," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1364–1390, 3rd Quart., 2015.

[16] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems," *Int. J. Netw. Manage.*, vol. 24, no. 4, pp. 221–234, Jul. 2014.

[17] V. Uceda, M. Rodríguez, J. Ramos, J. L. García-Dorado, and J. Aracil, "Selective capping of packet payloads at multi-Gb/s rates," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 6, pp. 1807–1818, Jun. 2016.

[18] Tcpdump & Libpcap. (2020). *Library for Network Traffic Capture: LibpCap*. [Online]. Available: http://www.tcpdump.org

[19] S. N. Z. Naqvi, S. Yfantidou, and E. Zimányi, "Time series databases and InfluxDB," Advanced Databases Studienarbeit, Université Libre de Bruxelles, Brussels, Belgium, 2017.

[20] N. Shah, D. Willick, and V. Mago, "A framework for social media data analytics using Elasticsearch and Kibana," *Wireless Netw.*, pp. 1–9, Dec. 2018.

[21] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2037–2064, 4th Quart., 2014.

[22] Y. Yang, Q. Cao, and H. Jiang, "EdgeDB: An efficient time-series database for edge computing," *IEEE Access*, vol. 7, pp. 142295–142307, 2019.

[23] K. Shafi, U. S. Ahmad, S. Nawab, W. K. Bhatti, S. A. Shad, Z. Hameed, T. Asif, and F. Shoaib, "Measuring performance through enterprise resource planning system implementation," *IEEE Access*, vol. 7, pp. 6691–6702, 2019.

[24] SAP. (2020). *SAP Business Suite*. [Online]. Available: https://www.sap.com/products/business-one.html

[25] HPCN. (2020). *Measurement Campaigns*. [Online]. Available: https://github.com/hpcn-uam/deepfda-experiments/tree/master/dataset
[26] D. Perdices, J. E. López de Vergara, and J. Ramos, "Deep-FDA: Using functional data analysis and neural networks to characterize network services time series," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 986–999, Mar. 2021.
[27] Grafana. (2020). *Beautiful Metric & Analytic Dashboards*. [Online]. Available: http://grafana.org/
[28] C. Vega, E. Miravalls-Sierra, G. Julían-Moreno, J. E. López de Vergara, E. Magaña, and J. Aracil, "On the design and performance evaluation of automatic traffic report generation systems with huge data volumes," *Int. J. Netw. Manage.*, vol. 28, no. 6, p. e2044, Nov. 2018.
[29] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: A deep learning approach for short-term traffic forecast," *IET Intell. Transp. Syst.*, vol. 11, no. 2, pp. 68–75, Mar. 2017.
[30] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.
[31] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand.*, Dec. 2013, pp. 273–278.
[32] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Feb. 2003.
[33] C. Olah. (2015). *Understanding LSTMs*. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural, Comput.*, vol. 9, no. 8, pp. 80–1735, Dec. 1997.
[35] C. Vega, J. Aracil, and E. Magaña, "KISS methodologies for network management and anomaly detection," in *Proc. 26th Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2018, pp. 181–186.
[36] S. Mao and F. Xiao, "Time series forecasting based on complex network analysis," *IEEE Access*, vol. 7, pp. 40220–40229, 2019.
[37] Sarkar, R. Bali, and T. Ghosh, *Hands-On Transfer Learning With Python: Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras*. Birmingham, U.K.: Packt Publishing, 2018.
[38] F. Chollet *et al.* (2020). *Keras*. [Online]. Available: https://keras.io
[39] HPCN. (2020). *Player of Network Anomalies*. [Online]. Available: https://github.com/jlgarciadorado/IncidencesPlayer
[40] J. Grossman, M. Grossman, and R. Katz, *The First Systems of Weighted Differential and Integral Calculus*. Rockport, MA, USA: Archimedes Foundation, 2006.
[41] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020.
[42] Z. Xuanxuan, "Multivariate linear regression analysis on online image study for IoT," *Cognit. Syst. Res.*, vol. 52, pp. 312–316, Dec. 2018.
[43] W. Zhang, Y. Wen, L. L. Lai, F. Liu, and R. Fan, "Electricity cost minimization for interruptible workload in datacenter servers," *IEEE Trans. Services Comput.*, vol. 13, no. 6, pp. 1059–1071, Nov./Dec. 2017.
[44] M. H. Lambert, *A Model for Common Operational Statistics*, document IETF Request For Comments 1857, Oct. 1995.
[45] T. Oetiker, "Monitoring your IT gear: The MRTG story," *IT Prof.*, vol. 3, no. 6, pp. 44–48, 2001.

DANIEL PERDICES received the B.Sc. degree (Hons.) in mathematics, the B.Sc. degree in computer science, the M.Sc. degree in mathematics and applications, and the M.Sc. degree in information and communications technologies from the Universidad Autónoma de Madrid (UAM), Spain, in 2018, 2019, and 2020, respectively, where he is currently pursuing the Ph.D. degree with the High Performance Computing and Networking (HPCN) Research Group. He is also an Assistant Researcher with the High Performance Computing and Networking (HPCN) Research Group, UAM, where he has received a four-year predoctoral fellowship by the Spanish Ministry of Science, Innovation, and Universities. He was a Research and Development Engineer with Naudit HPCN, Spain. His research interests include deep learning, statistics, mathematical modeling, network traffic analysis, and software defined networks.

JOSÉ LUIS GARCÍA-DORADO received the M.Sc. and Ph.D. degrees in computer and telecommunications engineering from the Universidad Autónoma de Madrid (UAM), Spain, in 2006 and 2010, respectively. Since 2005, he has been a member of the High Performance Computing and Networking (HPCN) Research Group, UAM. He was awarded a four-year predoctoral fellowship by the Ministry of Education of Spain, in 2007. He was a Visiting Scholar with the Telecommunication Networks Group, Politecnico di Torino, Italy, in 2010, the Internet Systems Laboratory, Purdue University, USA, in 2013, and the Faculty of Applied Science, Universidad Técnica del Norte, Ecuador, in 2014 and 2015. He is currently an Associate Professor with UAM. His research interests include analysis of Internet traffic: its management, modeling, and evolution.

JAVIER RAMOS received the M.Sc. degree in computer science and the Ph.D. degree in computer science and telecommunications from the Universidad Autónoma de Madrid, Spain, in 2008 and 2013, respectively. He was a Visiting Researcher with the Fraunhofer Institute for Open Communication Systems FOKUS, Germany, in 2012. He is currently an Associate Professor with the Universidad Autónoma de Madrid. His research interests include analysis of network traffic, quality of service, software-defined networks, and network function virtualization.

RODRIGO DE POOL received the double degree in mathematics and computer science from the Universidad Autónoma de Madrid (UAM), in 2020, where he is currently pursuing the M.Sc. degree in mathematics and applications. While studying for his degree, he enjoyed four grants from the Madrid Local Government for his excellent academic record and collaborated with Naudit HPCN, Spain. He actively participated in programming competitions reaching international level, in 2018 and 2020 (SWERC competition). He is holding a Severo Ochoa grant to do mathematical research at the Institute of Mathematical Sciences (ICMAT), UAM. His research interests include algebraic geometry and topology.

JAVIER ARACIL received the M.Sc. and Ph.D. degrees (Hons.) in telecommunications engineering from the Technical University of Madrid, in 1993 and 1995, respectively, and the five-year degree in mathematics from UNED, in 2009. He was awarded a Fulbright scholarship to pursue postdoctoral research at the University of California at Berkeley, Berkeley, CA, USA, in 1995. He was a Research Scholar with the Center for Advanced Telecommunications, Systems, and Services, University of Texas at Dallas, in 1998. He was an Associate Professor with the University of Cantabria and the Public University of Navarra. He is currently a Full Professor with UAM and a Founding Partner of the spin-off company Naudit HPCN. He has authored more than 100 papers in international conferences and journals. His research interests include optical networks and performance evaluation of communication networks.