View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Technical Disclosure Common

Technical Disclosure Commons

Defensive Publications Series

October 08, 2018

REAL-TIME MULTI-VARIATE MULTI-TIME-SCALE ANOMALY DETECTION SYSTEM FOR NEXT GENERATION NETWORKS

Antonio Nucci

Song Cui

John Garrett

Gurvinder Singh

Kenneth Croley

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

Nucci, Antonio; Cui, Song; Garrett, John; Singh, Gurvinder; and Croley, Kenneth, "REAL-TIME MULTI-VARIATE MULTI-TIME-SCALE ANOMALY DETECTION SYSTEM FOR NEXT GENERATION NETWORKS", Technical Disclosure Commons, (October 08, 2018)
https://www.tdcommons.org/dpubs_series/1579

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

REAL-TIME MULTI-VARIATE MULTI-TIME-SCALE ANOMALY DETECTION SYSTEM FOR NEXT GENERATION NETWORKS

AUTHORS:
Antonio Nucci
Song Cui
John Garrett
Gurvinder Singh
Kenneth Croley

ABSTRACT

Techniques are described herein for a real-time multi-variate, multi-scale, context-aware anomaly detection system. This system is built using concepts of edge/cloud distributed processing and orchestration.

DETAILED DESCRIPTION

By 2021 networks are forecast to grow by a factor of 75% (61,000 connected network devices) and corresponding incidents per hour will reach the order of 10,000 quotas. Information Technology (IT) providers have responded by delivering to the market a new generation of network devices which will extract more information about the health and status of each network device and export such telemetry at a higher frequency rate (evolving to a real-time push model to deliver network telemetry). While this will grant IT providers with fresher and richer data, next generation anomaly detection systems are called to address many technical challenges: how to effectively process massive amounts of data; how to analyze such data in streaming; how to design anomaly detection systems that can operate across many metrics of observation and multiple time-scales; how to introduce learning and intelligence into those next generation systems such that they can become smarter the longer they are used and exposed to domain-experts; etc.

Today, a typical enterprise comprises an average of 35,000 connected network devices which are managed by processing, on average, 750 Gigabytes of diagnostic data per day (e.g., syslogs, command line interface (CLI) data, configuration files, etc.). Usually such data is pulled by the IT provider after the occurrence of a network problem or customer escalation and used to precisely troubleshoot the issue, identify the root cause, and hence deploy a successful remediation. This process requires a long cycle (requires an average of five hours per incident), is executed by the IT provider on a very regular and frequent basis (enterprises reported in 2017 an average of 400 incidents per hour), and is very prone to human error (e.g., IT domain experts play a critical role in the overall troubleshooting process). While IT is already struggling today, next generation networks will make it even worse.

Next-generation anomaly detection systems are called to continuously ingest 70+ Terabytes of telemetry every hour (assuming a forecasted 61,000 router enterprise, with a network device generating an average of 20 Megabytes per minute). Clearly the delivery of such voluminous amounts of data to enterprise Data Centers (DCs) (or even worse, private/public clouds for analysis) will result in a high level of network congestion, prolonged analytical workflow completion times, and long remediation cycles. Next generation anomaly detection systems should adopt a distributed data processing model, meaning that some data processing should be executed as close as possible to the source of the data while other processing may still be executed in the enterprise DC or clouds. The correct distribution of processing workloads guarantees lighter data volumes to be exported to the DC and the cloud (no network congestion), shorter completion times for advanced machine learning analytics workloads (model training may use smaller amounts of data as input) and hence shorter remediation cycles (faster troubleshooting translates into faster deployment of fixes).

Next-generation network devices may generate very rich telemetry, which allows IT providers to properly monitor the correct behavior of network devices from many vantage points. For example, certain devices may export telemetry which includes details about the memory and Central Processing Unit (CPU) utilization of each device, comprising logical nodes, counters of packets traversing each device interface, rich statistics about power consumption and usage of its logical and physical components, etc. As a result, anomaly detection systems are called to generate more holistic profiles of network devices or the network-as-a-whole by modeling many distinct collected metrics together. More specifically, they need to profile every metric in isolation (like memory utilization on every logical node of the device) while discovering and modeling possible hidden and stealthy correlations across the distinct metrics (like memory utilization across logical nodes of the device).

Many problems affecting a network device do not manifest themselves with major deviations of a single metric but rather appear as smaller, hard-to-see deviations spread across many distinct metrics with strong temporal synchronicity. This is similar for entire networks. Consider as an example the most devastating Distributed Denial of Services (DDoS) attacks which target a network element causing resource exhaustion and starvation. The network element being targeted is attacked using many machines which send similar Internet Protocol (IP) packets (same size, same Transmission Control Protocol (TCP) flags set, etc.) with strong time synchronicity to starve resources at the victim endpoint. The packets are spread across many logical paths in the network which cross many interfaces of the many routers traversed toward their destination. Profiling a single metric such as a packet counter on a single interface of a single router would not trigger any alert. Conversely, correlating the same (or similar) IP packets across all router interfaces and across all routers along the logical path in the network would signify the active presence of the attack. Algorithms that generate behavioral models which take into account more than one time-series metric are called multi-variate. Furthermore, because the drifting from the normal behavior state happens at different frequencies (and hence at different time-scales such as minutes, hours, days, or weeks), it is imperative for anomaly detection systems to model and profile the behavior of each network device at different time-scales, from a micro-time-scale of minutes to a macro-time-scale of weeks. This property is called multi-timescale behavioral profiling.

It is imperative for IT providers to access detailed telemetry during the troubleshooting phase (which starts upon the generation of an alert). Today, this is achieved by exporting all collected telemetry to a central data storage infrastructure (e.g., a data lake) which makes the data accessible to the IT provider via standard Application Programming Interfaces (APIs). But with networks streaming over 80 Terabytes of telemetry every hour, this solution is clearly not feasible any longer. Hence, there remains the question of what can be done to provide the IT provider with the information needed to troubleshoot an alert while avoiding network congestion.

Although unsupervised anomaly detection systems are appealing for any IT organization for being less labor intensive than supervised systems, they are based on the assumption that any pattern that deviates from the learned normal patterns should be considered an anomaly. However, this assumption may not hold true because it is very difficult or impossible to define a normal event which takes all possible normal patterns/behaviors into account. More importantly, the boundary between normal and anomalous behaviors is often ambiguous, as in the case of an enterprise planned maintenance cycle. In addition, under realistic conditions, the same behavior could be a normal or an anomalous behavior under different conditions. As alerts are investigated by domain experts it is important for the anomaly detection system to assimilate the domain-expert knowledge to improve the learning process.

The system described herein (also referred to as "SQUID") distributes the data processing between the network itself and DC/cloud. The system embeds some data processing functionality directly into the network infrastructure itself (referred to as edge processing) to execute light processing functions such as collection, parsing, anomaly detection classifier at the device level, etc. The edge processing leverages dedicated compute resources available in the next generation network devices; for example, devices that dedicate up to four virtual CPUs (vCPUs) and 400 Gigabytes of Solid Disk Drive (SDD) for pure computing. The system also deploys some data processing functionality in the DC and/or cloud environments, (referred to as a cloud/DC processing environment) to execute more compute and memory intensive data processing. Examples of data intensive processing include the training of the machine and deep learning models. Furthermore, the cloud/DC processing environment may also correlate group alerts generated by each processing engine to uncover network-wide incidents which may affect more than one network device. Examples include DDoS attacks (wherein large numbers of machines generate an attack with strong time synchronicity toward a target endpoint), validation of correctness maintenance cycles (multiple devices may be power-cycled at approximately the same time), blast radius (multiple neighboring devices logically connected at approximately the same time), etc. The system connects the ecosystem of edge processing environments and the central processing environment via a secure bi-directional connectivity which is used to regularly distribute the learned / refreshed machine / deep learning models from the central to the edge processing environments.

The system described herein departs from more established single-variate anomaly detection systems, Exponential Weighted Moving Average (EWMA), Holt-Winters, etc., using a multi-variate multi-timescale anomaly detection sub-system. In contrast to well-established multi-variate anomaly detection algorithms such as the Auto Regressive Integrated Moving Average (ARIMA), which applies when the correlation among multi-features is linear and the input process is strictly ergodic (noise is assumed to be Gaussian distributed), this system is also capable of uncovering non-linear correlations while making no strict assumptions regarding the distribution of the underlying noise, which is the case for network devices. Unlike to more modern neural networks based anomaly detection systems, the system automates the selection of relevant feature for every input time series and for every time-scale and organizes the extracted features to retain the context of the particular logical entity of the network device / network that is applied (e.g., for a device, the system uses device (root level), logical node (level 1) or node runtime processes (level 2)). The system may use a four-layer custom designed Long Short Term Memory (LSTM) model to learn the behavior of the profiled network device.

This system deploys a special component in the edge processing environment which indexes the collected telemetry in real-time (elastic search instance or other). It manages the allocated memory with a circular-buffer implementation, which guarantees never to exceed a predefined memory quota. When an alert is generated by the anomaly detection classifier (or another component deployed in the edge processing environment), it is forwarded to the IT provider notification system. IT personnel may request the system to provide more detailed information about the alert via a query interface. As soon as the request is entered into the system, it is distributed to all edge processing agents. Each agent processes the request and if any relevant telemetry is found in a local cache, it is packaged, compressed and sent back to the central processing environment. All responses are aggregated at the central processing environment and presented together to the IT personnel. The sizing of the circular buffer for local indexing may be fully customizable. For example, for a device that generates 20+ Megabytes of telemetry every minute and is provisioned with 400 Gigabytes of SSD, the circular buffer may cache all data records collected for over 12 days before flushing the buffer and starting again. The current implementations force a regular flush-out/reset of the buffer every 24-hours and upload the content as a file into the central processing environment during pre-scheduled maintenance windows.

The system includes an annotation tool used by domain experts to annotate an alert after their troubleshooting. Users can tag/annotate an alert using a two-deep tagging tree. For example, users can use the first level to annotate the alert as a "true positive" or "false positive" and the second level to add more details, such as "true positive; planned" or "true positive; unplanned". The information entered into the annotation tool is then automatically used by the LSTM learning model to recalibrate the model and hence assimilate the new knowledge. As a result, over time the learning models are automatically refined by assimilating the domain-expert knowledge. The models become more precise in the categorization of the anomalies, which translates into a higher operational efficiency (e.g., shorter cycles for troubleshooting). The IT operational benefits are numerous. For example, the IT provider may ask the system to learn a special model for network device behavior during a maintenance window. In response, the system may group all devices with anomalies tagged as "true positive; planned" and generate a model that captures their specific behavior. This may be used to suppress the generation of future alerts (e.g., filtering based on new model), validate the correct execution of a planned maintenance cycle (e.g., planned maintenance on ten devices and observe ten affected devices), and/or automatically detect whether any anomaly occurring outside the planned maintenance window shows strong behavioral similarity with the behavior of the maintenance window (e.g., an employee unplugging a network device and forcing the device reboot). Similar use cases are also applicable.

Figure 1 below illustrates an example high-level logical design of an anomaly detection system for high-data-throughout networks. The system includes two logical processing environments: the edge processing and the central processing. The edge processing is compact and requires light compute resources from the hosting environment. This makes it ideal as a processing environment which can be embedded directly into next generation network devices. Of course, the edge processing can also be deployed elsewhere in any third-party servers placed close to the network infrastructure and/or sources of data, (e.g., a server-based dedicated collector). The central processing is usually deployed in compute environments which can offer more horsepower (e.g., enterprise DCs or clouds). Indeed, central processing is mostly used for regular generation, training, and recalibration of high-compute-intensive deep learning anomaly detection models which, when completed, are packaged as classifiers and dispatched to the edge processing environments for alert generation. This environment may also be used to host a network-wide classifier, similar to the one deployed in each edge processing but operating on all telemetry extracted from each edge processing module. The two processing environments are securely connected to exchange information and knowledge.

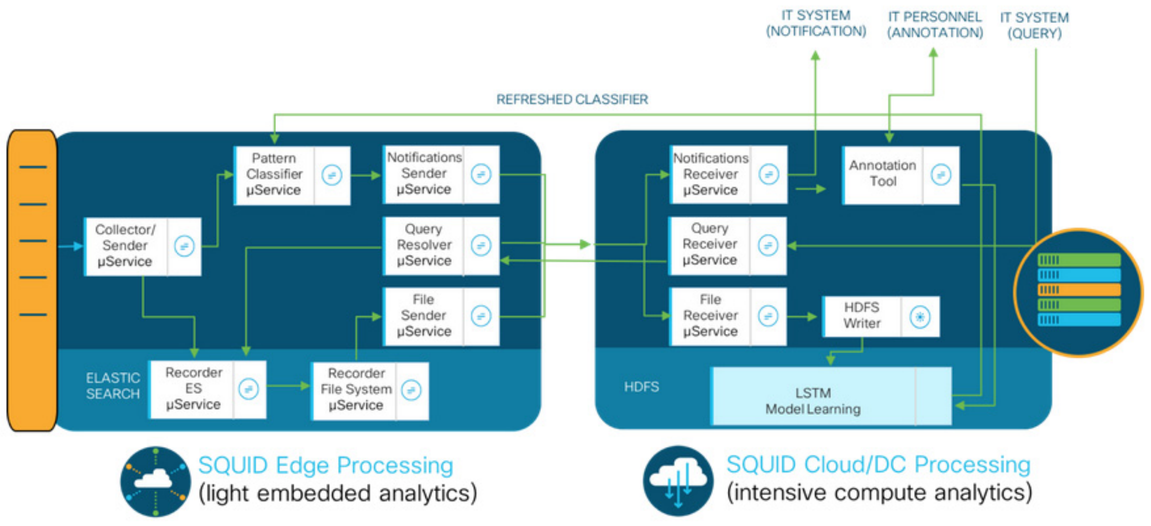


Figure 1. SQUID System View: Edge Processing and Central

The edge processing is comprised of the following functional components, which are referred in this patent application as micro-services. The Collector/Sender component continuously ingests the streaming telemetry generated by the network device and makes it available to two down-stream components, the pattern classifier and the recorded elastic search agents. The Pattern Classifier receives the data telemetry as an input and searches for abnormal behaviors by executing the latest learned model delivered by the central processing. As soon as an out-of-norm behavior is detected, the pattern classifier informs the notification sender microservice which packages the required information and delivers streams it to the central processing, where it is picked up by the notification receiver microservice and forwarded to the IT notification system via a Representational State Transfer (REST) API. The Recorder ES microservice is in charge of indexing the streaming data and locally persists in the elastic search instance. This microservice regularly monitors the memory usage and when approaching the pre-selected quota, it flushes the cached data into a file, compresses the file, and invokes the File Sender microservice to upload the file in the central processing system.

One other important functional component of the edge processing is the Query Resolver microservice which is invoked at the time of troubleshooting an alert. IT personnel can cause the system to recollect specific information related to a specific anomaly they have been investigating. As soon as the system receives the request, entered as a query in an abstracted language using the system interface, the system informs the Query Receiver microservice about the content of the request which in turns dispatch such request by initiating several internal requests to all the system edge processing units connected to the system. Every edge processing unit receives the internal-query and invokes the Query Resolver microservice which checks whether any metadata cached in the local elastic search index meets the criteria. If so, the query resolver packages the metadata and forwards back to the central unit. The Query Receiver deployed in the central unit recollects all the responses from the connected edge processing units and presents the discovered information using the same system interface used by the IT provider to enter the original request.

The central processing is comprised of three more functional components: the Hadoop Distributed File System (HDFS) Writer, the Anomaly Detection Model Learning microservice and the Annotation Tool. The HDFS Writer microservice is in charge of writing into the central processing HDFS infrastructure the file as received by each system edge processing. A new file is produced every time a system edge processing unit empties its local elastic search instance. The Anomaly Detection Model Learning microservice reads all the data persisted in the HDFS infrastructure and executes, at a predefined time schedule (via a scheduler), a pipeline of machine / deep learning jobs which generate a rich behavioral model for every network device under system watch. Every time a new model is generated, this module dispatches the most recent models to all connected edge processing units which start using them in their pattern classifier microservice. The system closed loop information flow ensures that all pattern classifiers always use the most current behavioral models and hence can closely adapt to absorb new legitimate network dynamics that otherwise (as in the case of open loop anomaly detection systems) could lead to the faulty generation of network anomalies.

Last, the Annotation Tool allows IT providers to annotate every anomalous event been triggered after proper troubleshooting and investigation. Using the annotation tool, the IT provider can approve an anomaly (i.e., the system will keep generating similar alerts in the future) or can decline an anomaly (i.e., the system will re-adjust its internal LSTM weights and stop generating alerts for similar behaviors). Furthermore, the IT provider can use the tool to add extra tags for a true anomaly to improve their operational efficiency. For example, the IT provider can annotate a true anomaly as “maintenance window” and use this information to either detect unplanned events (e.g., an employee forcing a power cycle of a network device outside the planned maintenance window) or to validate the correctness of execution of their maintenance cycle (e.g., ten devices were supposed to be power-cycled during the maintenance window but thirty devices were forced to be power-cycled).

There are a variety of models that can be used for anomaly detection in time-series data. For example, algorithms such as Holt-Winters, ARIMA, and Hidden Markov Chain models all capture temporal dynamics of a time series and produce a generative distribution for predicting the ranges of future values. However, these models cannot be used for network device signals. Each model comes with its own assumptions that affect the type of signal that can be fitted well with it. For example, Holt-Winters, Cumulative Sum (CUSUM) and EWMA models work only with one time-series as an input and use pre-specified time-windows over which the baseline is computed. The results greatly depend on the selection of this time window, which translates into how much memory of the past those algorithms retain. ARIMA models can work with multiple time-series as an input but assume the time series to be ergodic, meaning that the noise model strictly follows a Gaussian distribution. A Hidden Markov Chain model makes an assumption on the memory of the process that a current device state depends only on the previous state, even though this is not the case because external perturbations may cause the device to drift into a different state at any time.

Very recently, next generation anomaly detection systems have become productized using machine learning algorithms that overcome the aforementioned challenges. One system is designed to detect real-time anomalies, more specifically sudden drops in volumes of data streams, received by their industry partners. This system is based on neural networks, and more specifically a customized version of the Deep Neural Network (DNN) Regressor model which belongs to the DNN family of algorithms. This module operates on each individual type of data stream (each stream considered independently from the others) and uses a static set of features (i.e., specific features for specific types of data streams) to train the learning model. The feature engineering process is manual and requires domain experts to define the set of relevant features to be used for the learning model. Even though this system belongs to the category of multi-variate anomaly detection systems (multiple time-series analyzed at the same time) it is not capable of modeling possible cross-correlations between the (e.g., fourteen) different types of data streams received as an input. Moreover, this system works on a pre-selected time window and hence is only capable of identifying behavioral anomalies which manifest themselves at the pre-specified time scale.

A different multi-variate anomaly detection system based on LSTM has been proposed. This system is a true multi-variate anomaly detection system in the sense that is capable of profiling each time-series in isolation as well as extracting and modeling the underlying cross-correlations which may exist across the different input time series. The system is also capable of automatically selecting relevant features from the input data using a feature extraction module. However, the system has been designed to operate on one single time scale (rolling window of fixed size) and hence is not capable of generating models capturing the behavior of the profiled entity across different time-scales. Though the system proved to be very powerful to detect extreme events during high variance segments, it is not directly applicable to the specific problem at hand, which calls to automate the feature extraction across many distinct time-scales of operation. This is a major difference because a relevant feature for a time-scale A may not be relevant any longer for a different time-scale B. For example, features such as the average number of packets at the device interface level may be a relevant feature for time-scales of one-hour or less but not relevant beyond that since the packet counter time series may start exhibiting non-linear behavior.

Furthermore, no pre-existing systems have addressed a specific challenge that is specific to the problem of the inherent structure of collected network telemetry. For example, a single physical network device may host multiple logical nodes, and each logical node may execute different runtime processes at different times. For example, the environmental temperature or power utilization metrics are relevant to profile the overall device behavior, while CPU and memory utilization may be relevant at the logical node and process levels.

To address these limitations, the system described herein comprises three functional components: the feature generation module which automatically computes features by applying a cascade of mathematical operators when considering a set of different time-scales; a Hierarchical Feature Data Structure which organizes the computed features based on three types of entities, which are device (root level), logical node (first level) and processes (second level); and a four-layer LSTM model which automatically selects relevant features from the set of available features to produce a reliable forecasting model.

Figure 2 below illustrates an example logical architecture of an LSTM-based multi-variate multi-scale anomaly detection system. The learning model at the top of Figure 2 receives a set of time series (e.g., memory and CPU utilization every sixty seconds for each logical node comprising a network device), computes a large set of telemetry features by applying a cascade of mathematical functions operating at different temporal aggregation time-windows (time-scale) and automatically selects the subset of relevant features for accurately modeling each time-series in isolation and possible cross-time-series correlation (via a four-layers LSTM implementation). The system multi-stage LSTM component automatically promotes specific features which are elected as important for the forecasting model and expire/decay other features which do not add any information to the forecasting model. The output of the system learning model is a forecast model which is then used as an anomaly detection classifier that receives real-time telemetry as an input and outputs anomaly notifications.

The system classifier at the bottom of Figure 2 is deployed in each of the edge processing engines. Every time a new LSTM anomaly detection model is made available by the system learning module component, it is dispatched to all system edge processing engines which use the refreshed model to reflect the most current behavior of the network device. The LSTM model acts on the received telemetry time-series and computes the corresponding residuals metrics which capture the deviations between the new data ingested and their expected forecasted behavior. If the residuals are larger than a pre-specified internal threshold, the classifier labels the data point as anomalous; otherwise it labels it as normal. To avoid generating alerts based on a single data point deviation from the forecasted model, the system classifier is equipped with a short memory buffer that maintains memory of the state of each time-series over the last M ingested data points. The classifier generates an alert if and only if N (N<=M) of the M observed data points have been tagged as anomalous. The memory buffer (M) and triggering mechanism (N) may be fully configurable by the user.

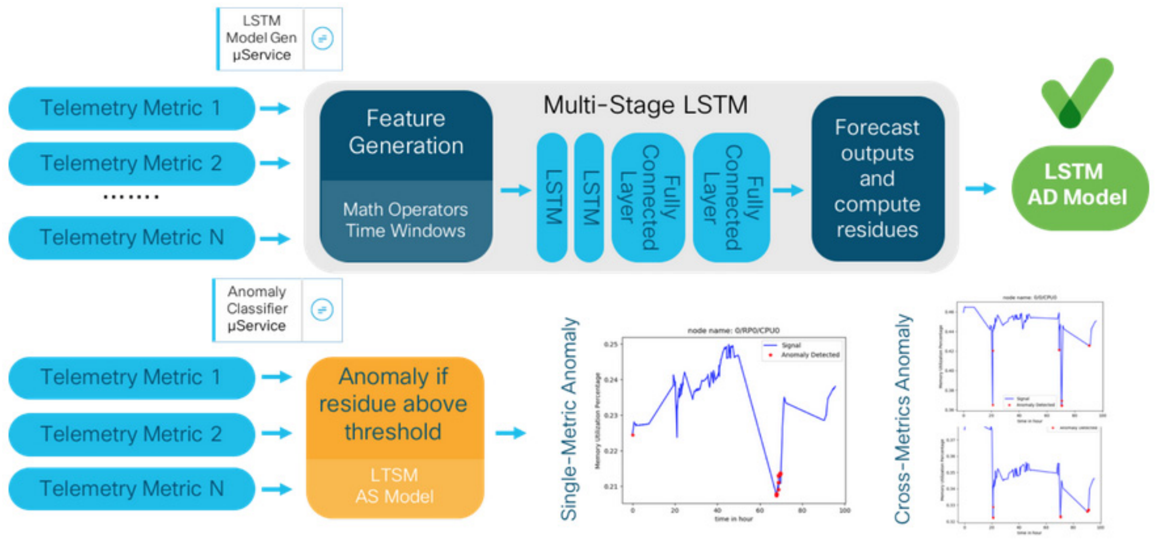


Figure 2. SQUID Multi-Variate Multi-Scale Anomaly Detection System using LSTM.

The system is equipped with a flexible and programmable feature generation module which addresses three common problems of LSTM models: which features can best represent the input data telemetry; which features should be selected as relevant for a given time-scale; and how the features should be combined/processed together to drive the generation of a single learning model.

The system feature generation module automates this process by cascading three functionalities.

The system uses a set of mathematical functions which are applied on the input data. Examples of mathematical functions include MAX, MIN, AVERAGE, VARIANCE, and PERCENTILE as well as advanced operators such as LINEARITY, CURVATURE, INFORMATION THEORY RELATIVE UNCERTAINTY, etc. For example, mathematical operators like LINEARITY assess whether any input time-series is linear versus non-linear for a given time-scale, or Gaussian versus non-Gaussian, and so-on. It is indeed well-known that packet counter time-series tend to be linear at a time-scale of one hour or less but then become non-linear.

Operators such as CURVATURE extract the underlying trend embedded in a time-series (e.g., linear), sudden shifts, or seasonality patterns. In the example of packet counters, the time-series exhibit strong seasonality, daily, weekly, and monthly trends. Operators such as RELATIVE UNCERTAINTY process the empirical distribution of the observed metrics for a given time-window and automatically extract its underlying shape and shape-shifts over time. In this example, a spike up in the distribution of the number of source IP address which are actively communicating to the same destination IP using similar packet sizes may be symptomatic of an active DDoS attacks against the target destination IP. If the DDoS attack is rapid, then this pattern may be observable at a small time-scale (e.g., minutes). Conversely, if the attack starts low and grows over time, the pattern may emerge only when aggregating at a larger time scale (e.g., hours).

All mathematical operators extract different insights from each time-series but these insights are relevant only within the context of the time window over which they are computed. The list of time windows driving the feature temporal modeling are provided by the math operator's time-window configuration file. The system modularity (i.e., abstraction between mathematical functions and time windows) allows users to easily add new mathematical operators without touching the time windows and vice-versa. The system computes the feature list by applying each listed mathematical function using each pre-specified time window. Finally, all extracted features (pairs of functions and time window for each input time-series) are organized in a hierarchy model which preserves the contextualization of relevant features with the logical entities being profiled: root, nodes, and processes.

The forecasting model contains a multi-layer LSTM deep learning model. Once the features are generated, they are sent to a multi-layer LSTM based forecasting model to predict the current telemetry values for all telemetric values. The multi-layer LSTM model, one of the deep learning models, utilizes a cascade of neural network layers to model the complex temporal correlations among single telemetry time-series and cross-correlations among all telemetry time-series in the system. This model is trained to learn the multi-scale behavior of the network device. The model may then be used to generate anomalies when a deviation between current telemetry and learned models are observed. The inputs to the four-layer LSTM model are the features generated by the Feature Engineering module.

There mathematical models suitable for describing the relationships among the four elements within an LSTM layer. LSTM is suitable for forecasting time-series signals as the input gate. The forget gate controls how the cell should memorize "short term" and "long term" patterns in the past. The output gate determines how the information stored in the cell can be used to forecast/predict the current value in the time-series.

In multi-layer LSTM networks, single LSTM layers are stacked together which makes the model deeper. The increased complexity enables the model to better learn the inherent nonlinear patterns in the signals. Each layer processes some parts of the task to solve and pass on to the next layer. Figure 3 below illustrates an example model architecture for this anomaly detection system.

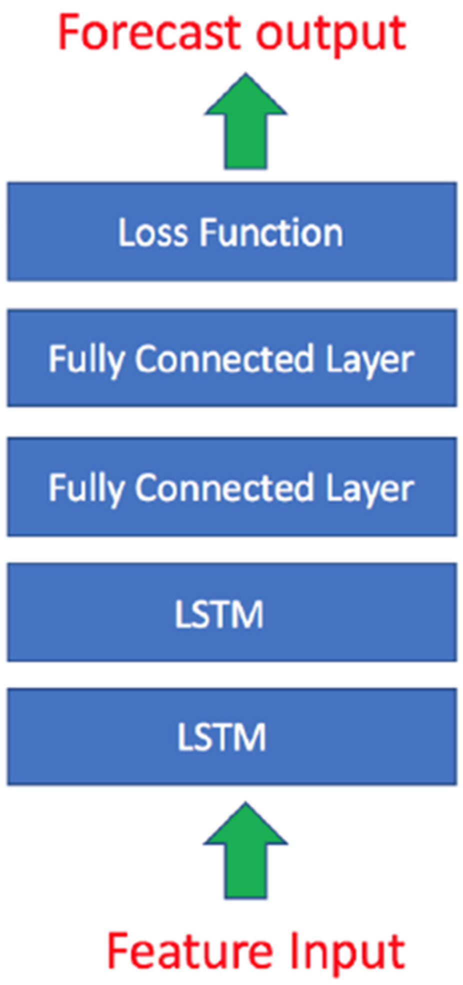


Figure 3 Multi-stage LSTM Neural Network

A fully connected layer is a layer where every neuron is connected to every neuron of the next layer. The loss function specifies how the model training penalizes the forecasting value and the true value. In this anomaly detection system, "mean-squared-error" is used as the internal loss function. The number of LSTM layers and fully connected layers may vary from network to network. In general, more layers should be used for more complex network devices (richer data telemetry), or to cover larger network deployments, more complex temporal-spatial correlation across different network devices may be determined.

After the forecasting model is trained, the anomalies can be detected by comparing the current telemetry values and the values predicted by the forecasting model. If the incoming telemetry data is consistent to what observed in the past, then it is labeled "normal" Conversely, if the incoming telemetry data behaves differently from what was observed in the past, it is labeled "anomalous" In the latter case, the residual values, which capture the difference between the current values and the predicted values, can be used to quantify the divergence and hence the severity of the alert to be generated. An internal threshold is pre-selected on the value of the residuals to label a data point as normal or anomalous. The system uses a memory buffer to minimize the number of alerts being generated and to avoid generating alerts which are related by one single data point. Indeed, only if M out of the last N data points are labeled anomalous, the system generates a single alert.

Provided are results used when applying the system to detect anomalies affecting one single network device in isolation or more than one network device in the network. To that end, Figure 4 illustrates a system Annotation Tool and how it is actually used as part of the troubleshooting and learning life-cycle. As soon as the classifier (residing in the edge processing embedded in each network device) generates an alert (Figure 4.a), the alert is propagated by system (from edge processing to central processing via secure connectivity) to the Annotation Tool (Figure 4.b). IT personnel can use the Query/Response functionality to retrieve the needed information to properly investigate the problem and land to a final conclusion.

Without the Annotation Tool, the system would not have any way to assimilate back such knowledge that would be lost. As soon as the user annotates the anomaly (false positive, true positive, true positive but maintenance window, etc.), the system takes this information and sends it back to the multi-stage LSTM model in charge of learning the models (Figure 4.c). The LSTM model interprets the annotation and recalibrates the internal weights (LSTM cell states) to rapidly forget the pattern by multiplying the forget gate by 1 (false positive) or to keep a long memory of it (true positive). The learned model is then dispatched to all the edge processing engines, which receive the new model and update their local classifiers accordingly (Figure 4.d). The system Annotation Tool supports two-level tagging but may be generalized to enable N-level tagging.

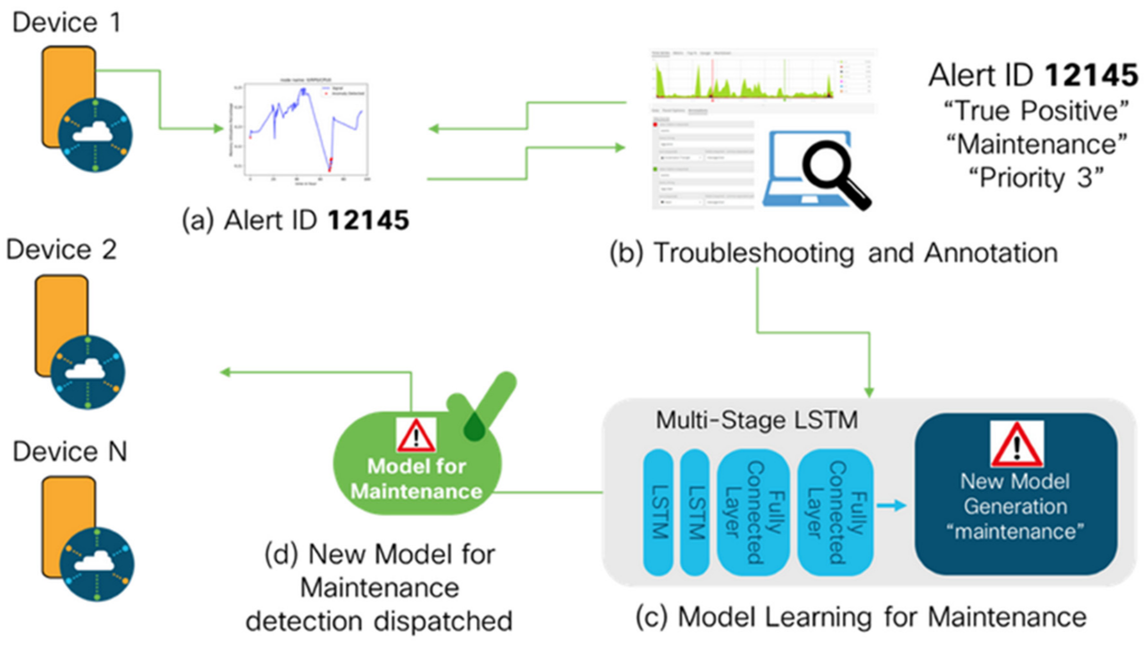


Figure 4 SQUID Annotation Tool and Anomaly Life-cycle

Also provided are live alerts generated by the system as a proof-of-concept. In this specific deployment, a network comprised of ten routers was monitored, each equipped with the system edge processing. All edge processing are secured and bi-directionally connected to the system central processing. In this specific scenario, each edge processing has been programmed to collect and profile metrics related to memory and CPU utilization of all logical nodes of the network device as well as dynamics of the packet counters for every single interface.

Figure 5 below illustrates a relatively simplistic scenario where one specific logical node of a network router experienced a sudden increase of 20% in memory utilization. The system triggers four alerts at the exact times of a memory utilization spike (red dots).

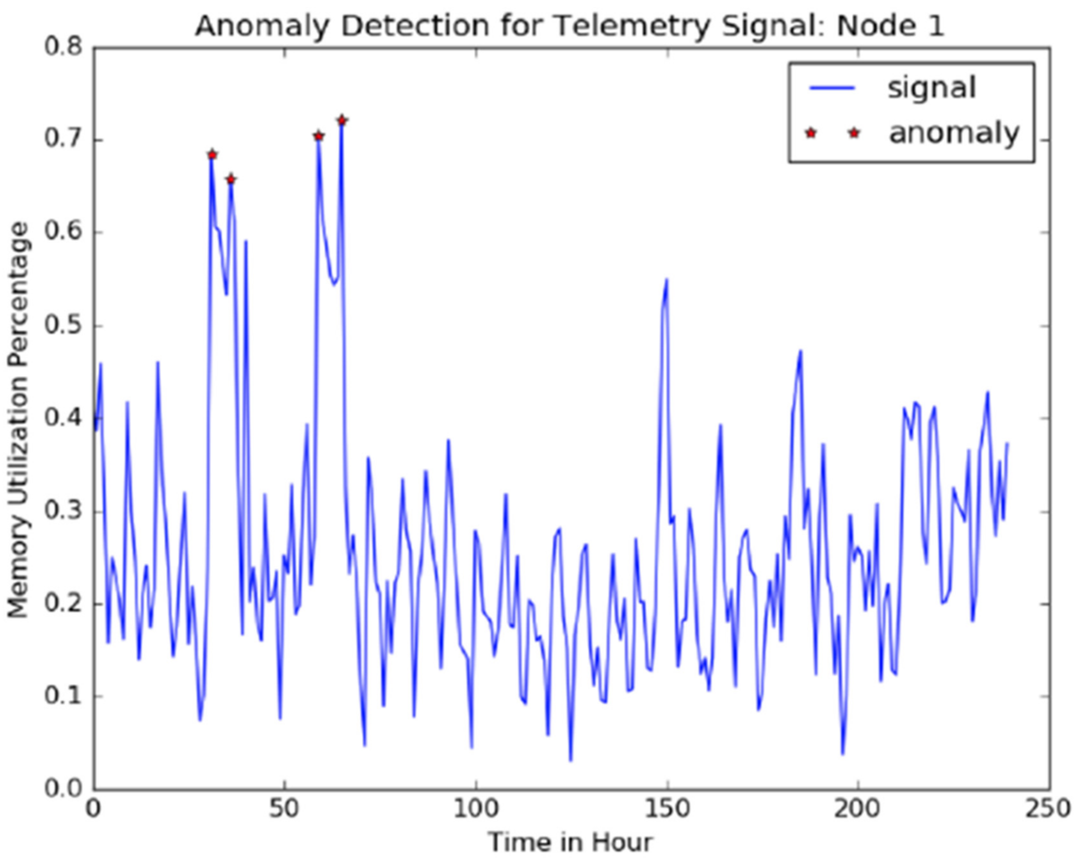
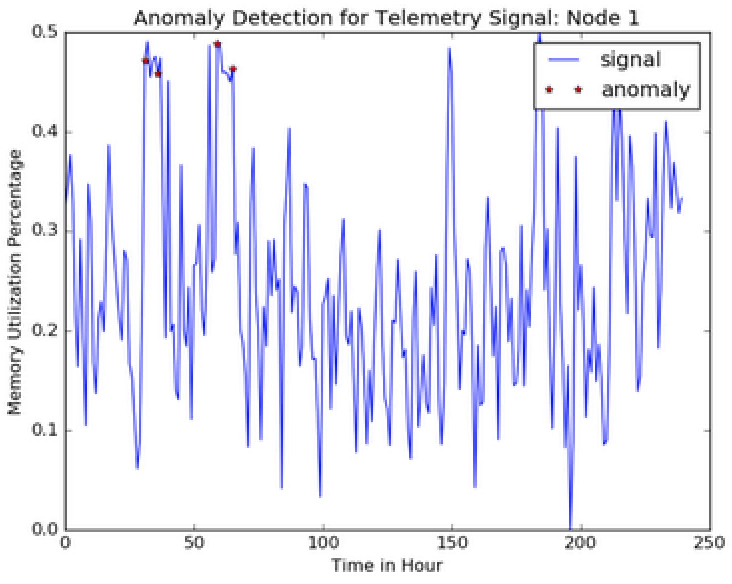


Figure 5 ALERT: One Device - One Logical Node (Node 1) - One Metric Memory Utilization

Figure 6 below illustrates the potential of the system when it leverage its model for cross-correlation of time-series and logical entities to promptly detect more subtle anomalies like minor spikes in memory utilization for two of the logical nodes (10%) but appearing with strong temporal synchronicity. The system labels the four abnormal times on both the logical nodes / memory utilization with red dots.



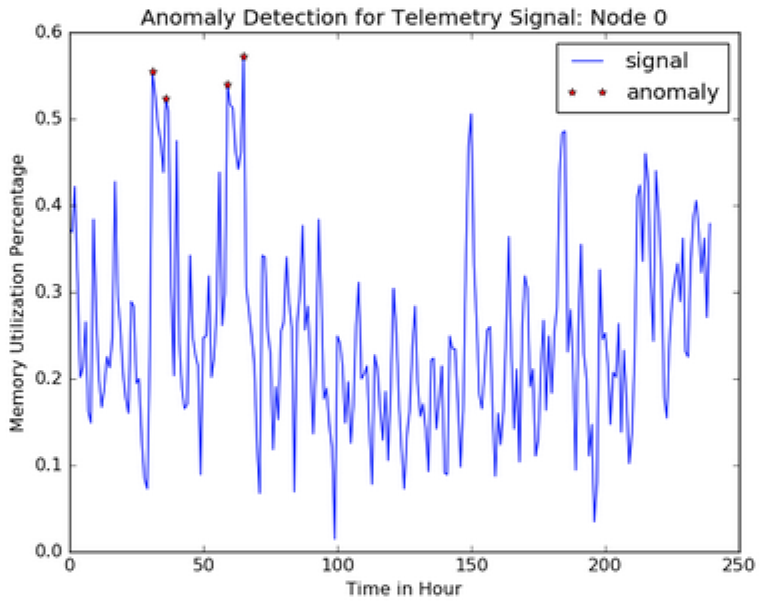


Figure 6. ALERT: One Device - Two Logical Nodes (Node 1 on the top, Node 0 on the bottom) - Two metrics Memory Utilization

The system is a general-purpose anomaly detection that is capable of analyzing different types of telemetries such as number of CPU processes, memory utilization, and packet counts at the same time. Thus the anomalies flagged by the system are not limited to abnormal traffic behaviors. The feature generation module in the system is designed to extract a large number of time-series behaviors at different time scales regardless of any specific telemetry type. For example, there is one feature measuring the degree of fluctuation of the incoming telemetric signal by computing standard deviation within different time windows. Obviously, this feature is applicable to various kinds of telemetries and not limited to a specific telemetric type. With the vast number of features generated at the network, device, and node levels, and a single deep learning model (multi-layer LSTMs), the system is capable of finding complex correlations and insights both inside a single device and cross different devices.

The system is aimed at detecting anomalies at both the device and network level. A hierarchical feature data structure is provided to organize computed features at network, device, and node levels. For example, the system can compute the number of CPU processes for one device and the whole network as two different features. Features at different hierarchy levels describe behaviors at different levels. In some scenarios, the number of CPU processes running at individual devices might have small deviations from normal behavior and may not be obvious to detect. However, the number of CPU processes running at the network may be significantly abnormal. This is possible for large-scale networks with thousands of devices. The feature engineering design makes the system capable of capturing anomalies beyond the device level. The system anomalies from telemetry data can be based on traffic counters, component level features, and system wide features combined into a single feed.

Deep learning functionality is only one part of the system. The system is a multi-variate and multi-scale anomaly detection system customized for a large-scale telemetry network.

The machine learning part may include two steps. The first step involves the feature generation module. It automatically computes features by applying a cascade of mathematical operators when considering a set of different time-scales. A hierarchical feature data structure organizes the computed features based on three types of entities, which are device (root level), logical node (first level), and processes (second level). This design is extremely important to power the SQUID to have the capability to detect anomalies from multiple telemetries at different time scales. Different mathematical operators aim to model different time-series signal behaviors. For example, the standard deviation models the degree of fluctuations. These mathematical functions are computed at different hierarchical levels of the network such as network level and device level. This design helps the system to detect abnormal behavior at different hierarchical levels. In addition, different telemetry signals have different seasonality cycles and the seasonality cycles may change from time to time. For this reason, the mathematical functions may be computed at different timescales as well. The system has a list of time windows with both default window lengths and configurable window lengths.

The second step is a four-layer LSTM machine learning model which is capable of learning the complex and hidden nonlinear cross-telemetric and cross-device behaviors in multiple time-scales and detecting the anomalies if these behaviors are abnormal. These capabilities are extremely important for large-scale telemetry networks where thousands of devices and millions of metrics with different seasonality cycles are highly connected and need to be monitored. This could not be achieved by ARIMA or LSTM alone.

The system can be used to analyze a single network protocol at all levels. For example, Open Shortest Path First (OSPF) packets between devices, OSPF processes on each devices in the OSPF domain, and OSPF interactions between nodes in the system may all be analyzed in the context of the full OSPF domain.

The system is capable of taking multiple categories in different dimensions. For example, the system is capable of taking human annotations “true positive (TP) / false positive (FP); planned (P) / unplanned (U)”. There may be four corresponding categories in total instead of “yes or no”. The system is capable of learning multi-category classifications by designing an appropriate loss-function in multi-layer LSTMs (Figure 3). As an example, a “soft-max” function may be used as the loss function and it has been proven that this loss function is effective in classification problems with more than 1000 categories.

In one example involving OSPF, planned maintenance windows that impact multiple devices may be identified as expected anomalies. Annotation which indicates the type of activity may be used to identify the behavior of the system (e.g., the entire OSPF domain) while the maintenance activity is underway, and create a classification for this activity. This learned pattern becomes a new class. Unauthorized maintenance activity, or an outage that matches the pattern, may be identified as an anomaly based on this knowledge, and the known matching activity may be offered as a root cause.

The system is not simply about distributing data processing, but is also about having new capabilities based on the data distribution. The training of the models occurs at the

DC/cloud to learn the complex behavior inside one device, learn the behaviors across multiple devices at a network level to detect network level anomalies, and provide group alerts. For real-time anomaly detection inference, the system is designed to run at the edge. The whole pipeline including the data parsing, feature generation, and model inference is lightweight and may run at remote devices/nodes. The edge and central processing is connected bi-directionally to enable regular model training updates and deployment.

In summary, techniques are described herein for a real-time multi-variate, multi-scale, context-aware anomaly detection system. This system is built using concepts of edge/cloud distributed processing and orchestration.