Early warning to resource depletion on networking devices

Elia Azar*
*École polytechnique
elia.azar@polytechnique.edu

***Abstract*—A networking device that runs out of resources, e.g. memory, TCAM and etc., becomes dysfunctional and can severely impact the business it supports. Device fabricators put in place various protection mechanisms. Some simply limits the max utilisation by killing/restarting jobs exceeding a manually tuned threshold. Some are more preventive by sending early warnings once the total consumption exceeds a critical point, that is again often arbitrarily defined by a domain expert. One major drawback of these known approaches is that they do not provide information on the moment when the concerned resources may run out, nor hint on the contributors to potential depletion, making it hard for users to design according countermeasures. Further, with ill-tuned thresholds, the warnings messages could be overwhelmingly frequent and of few relevance and lose users' trust and attention.**

In this paper, we aim at providing a preventive solution that is 1) data-driven so as to be adaptive and depend less on expert knowledge; 2) capable of hinting and updating the depletion moment with very few warning messages; 3) resources efficient such that the solution may run on networking devices.

We defined a set of criteria to benchmark the performance of early warning systems for resource monitoring. Data from 10 operational Cisco devices over 5 months are collected and used in testing. The proposed scheme outperforms a threshold-based Cisco solution from multiple aspects. More precisely, all 165 resource depletions are warned in avg. 2 hours and a half in advance. The warned depletion time is in avg. 42 minutes away from the actual occurrences.

***Index Terms*—Resource monitoring, warning generator, resource depletion, time-series forecasting**

I. Introduction

Multiple types of resources on a networking device (memory, TCAM, disk space) have limited resource provision. Not having enough of these resources available may cause unexpected device-local or system-wide consequences. For example, attempt on adding a new ACL (Access-Control list) that can not be aggregated to existing ones, when the TCAM is already full, could cause some traffic (directly concerned by this ACL or by other ACLs) to be dropped without explicit notice to network operator or end user. Better software engineering alleviates the consequence of such happenings. Nonetheless, it is of operational value to: learn any tendency in approaching the bottleneck of multiple essential resources, such that prevention may be taken, for example dimensioning, bug fixing, graceful device reboot investigation, and the like.

By looking at how the utilization of these resources evolve over a relatively short period of time (minutes to hours scale), we shall notice the utilization is changing, either increasing / decreasing continuously or in an oscillating fashion. Moving ahead of time from such observations, we shall end up always with conclusions as such: memory running out in 5 days or never depleting. Actually, it seldom continues this way, as for example, the allocated memory can be released in a big chunk way before the 5-day estimation thanks to scheduled garbage collection. On the other hand, the possibility also exists that a potential memory leak shall steadily drive the system to the border of failure. Eventually, we may ask, how may one tell whether the current trend is representing a normal system dynamic or is rather mixed up with a tendency to evolve towards a resource depletion ?

In this paper, the main focus is to alert the network administrator by generating early warnings before resource depletion, thereby giving him time to take appropriate countermeasures.

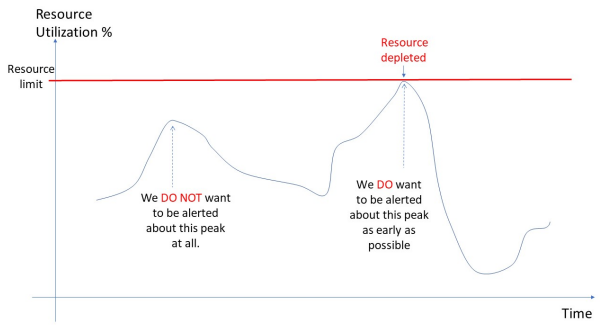

Figure 1. warning generator

Fig.1 illustrates that we don't want to be alerted about the first peak because it does not reach the resource cap, whereas we want to be alerted as early as possible about the second one in order to take appropriate countermeasures to mitigate the depletion.

The resource studied in this paper is memory. Memory is a resource that has data plane and control plane requirements. Memory is required for information such as routing tables, ARP tables, and other data structures. When devices run out of memory, some operations on the device can fail. The operation could affect control plane processes or data plane processes, depending on the situation. If control plane processes fail, the entire network can degrade. For example, this can happen when extra memory is required for routing convergence.

Memory leaks are static or dynamic allocations of memory that do not serve any useful purpose. In other words, when

a router starts a process, that process can allocate a block of memory. When the process completes, the process should return its allocated memory to the router's pool of memory. If not all allocated memory is returned to the router's main memory pool, a memory leak occurs.

Such a condition usually results from a bug in the Cisco IOS version running on the router, requiring an upgrade of the router's Cisco IOS image. A memory-allocation failure (which produces a MALLOCFAIL error message) occurs when a process attempts to allocate a block of memory and fails to do so. One common cause for a MALLOCFAIL error is a security issue. For example, a virus or a worm that has infested the network can result in a MALLOCFAIL error. Alternatively, a MALLOCFAIL error might result from a bug in the router's version of Cisco IOS [8].

Three different categories of solution exist for resource depletion.

The first one is the post-event category, it is not preventive and may kill non relevant processes. Resource depletion occurs when no more resource is available. The maximum resource utilization should be equal to 100%, based on the fact that when we have a given resource, we want to be able to use all of it, and not just a chunk, but the value set for the resource limit could vary from one system to another. For example, during manufacturing, the engineers could set the resource limit to a lower value, say 90%, forcing the device to take countermeasures like clearing file caches, restarting running processes and potentially reloading the node when this limit is exceeded in order to free the monitored resource. Watchdog [17] is a daemon / subsystem used to monitor the basic health of a machine. If something goes wrong, such as a crashing program overloading the CPU, or no more free memory on the system, watchdog can safely reboot the machine, allowing our service to keep on going. Watchdog is careful about the reboot. When it reboots the machine, it does so in a controlled matter. No sudden reboot. It kills processes, syncs the disks, unmounts the file-systems, and otherwise does a full reboot like the system normally would. For instance, Watchdog is constantly monitoring memory utilization and will take action when these thresholds are exceeded.

The second one is the threshold-based method. It is a solution deployed in production with the purpose of sending early warnings before resource depletion, but this method require manual threshold setting that is not adaptive.

The third one is the forecasting-based approach where time-series forecasting is used in order to predict the potential next values the monitored resource will go through. Several research studies related to resource monitoring and prediction were done and presented in the following section. Nevertheless, a gap exists between forecasting and early warning, hence the conception of this paper.

The threshold-based method is available in Embedded Resource Manager, ERM, in IOS-XE [2]. This approach requires manual setting of thresholds, as described in Section IV, and does not provide any information regarding the time at which the resource limit will be reached. In addition, this method is also recused due to the number of warnings falsely generated.

For that reason, this paper tackles a new approach based on forecasting. The gap between forecasting and early warning generation is closed by introducing a new mechanism that is able to generate meaningful warnings based on the last predicted values.

A benchmark system was implemented in order to compare our proposition with a threshold-based system also capable of generation warning messages (ERM).

The results seen in section VII prove that the solution proposed is able to make up for the shortcomings of the threshold-based method by predicting when the resource limit will be reached and by reducing the number of falsely generated alarms. The benchmark was done on data gathered from multiple real devices over five months.

The remainder of this paper is organised as follows. Section II introduces the related work while Section III discusses the benchmark criteria. Section IV formalises the state of the art and its limitations. Section V discusses the forecasting-based approach and section VI presents the associated experimental setup and evaluation. Section VII discusses the results obtained, discussion and future work will be mentioned in Section VIII, and finally, Section IX concludes this paper.

II. Related work

Overall, the focus of those works is on improving the accuracy of short-term forecasts. In contrast, in this work, the focus is on the usability of the time-series analysis outcome. As of today, a lot of algorithms and methods exist for time-series forecasting, which is being used in several domains, but the cornerstone of this paper is the analysis of the set of predictions in order to send early warnings to resource depletion in network devices. In this context, this paper is novel to this area of work.

Paper [14] shows that machine learning techniques can be successfully used to improve statistical approach for memory leak detection. Two learning algorithms (C4.5 and PART) bring improvement in leak detection quality. It also presents the challenges which had to be solved, method that was used to generate features for supervised learning and the results of the corresponding experiments. Classification is used to detect memory leaks therefore deviating from our perspective.

The work in [4] presents AUGURY, an application for the analysis of monitoring data from computers, servers or cloud infrastructures. The analysis is based on the extraction of patterns and trends from historical data, using elements of time-series analysis. The purpose of AUGURY is to aid a server administrator by forecasting the behaviour and resource usage of specific applications.

In [18] the authors study the effect of predictions on dynamic resource allocation on virtualized servers running enterprise applications. They used three different prediction algorithms based in a standard auto-Regressive (AR) model, a combined ANOVA-AR model, as well as multi-pulse (MP) model. The experiments were done on CPU utilization in-

formation from a collection of 48 servers hosting enterprise applications.

In [19], the authors present PRACTISE, a neural network based framework that can efficiently predict future loads, peak loads, and their timing. Extensive experimentation was done on traces from IBM data centers. They analyzed workload traces from production data centers and focused on their VM usage patterns of CPU, memory, disk, and network bandwidth.

The research studies done in [4], [19] and [18], do not fit the problem encountered in this paper due to the gap between forecasting and warning generation already mentioned in the previous section. The focus is on the usability of the time-series analysis outcome and not on the forecasting algorithms deployed.

Finally, as of today, the solution to sending early warnings to resource depletion is a Cisco tool that was implemented in Embedded Resource Manager, ERM [2]. This tool uses the threshold-based approach in order to send warnings. Details on how this solution works will be given in section IV.

III. Benchmark criteria

In this paper, the main focus is sending early warnings before resource depletion, this is why work on several aspects has to be done.

First, we have the **Precision**, which means that we are trying to differentiate between meaningful and non-meaningful warnings, or in other words, the difference between True Positive and a False Positive. A warning will be labeled as True Positive if it was able to successfully predict the happening of a depletion, otherwise, it will be labeled as False Positive if the warning was not followed by a depletion, meaning that the warning was generated in vain, it was a false alarm. Next, we have the **Recall**, taking into account if a depletion has been warned in advance or not, and highlights the difference between True Positives and False Negatives. In this case, an event is labeled as False Negative if depletion is reached and no warning was sent in advance about the happening of this depletion. Then, we have the **Earliness** being the distance in time between the time a warning was generated and the time depletion is reached. For example, being warned 4 hours in advance is better than 10 minutes, because this way, appropriate countermeasures can be taken on time. We also have the **Closeness** being the distance in time between the time depletion was predicted to happen and the time depletion is attained. In this case, the smaller the value of the $Closeness$ the better, and if the $Closeness$ is equal to zero, this means that we perfectly predicted when depletion will happen. Finally, we have **Resource Consumption**. This metric checks if the running method is consuming a lot of ram and cpu.

We are trying to find the optimal value for all these aspects, knowing that a trade-off will take place.

IV. Embedded Resource Manager

The threshold-based method is the common approach for resource monitoring nowadays. Embedded Resource Manager, ERM [2], is a Cisco tool based on setting thresholds that allows us to monitor internal system resource utilization such as buffer, memory and cpu.

The ERM framework tracks resource utilization and resource depletion by monitoring finite resources. Support for monitoring CPU, buffer, and memory utilization at a global or IOS-process level is available.

The ERM framework provides a mechanism to send notifications whenever the specified threshold values are exceeded by any resource user. This notification helps network administrators diagnose any CPU, buffer, and memory utilization issues.

The ERM architecture is illustrated in the figure below.

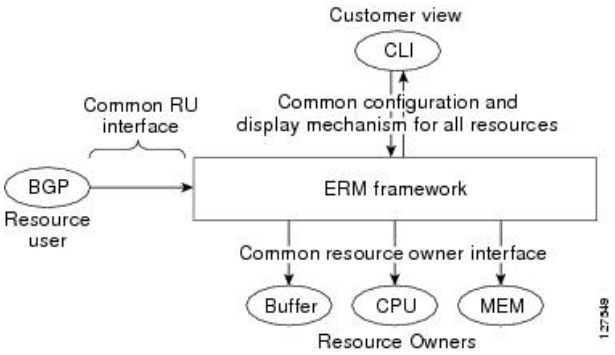

Figure 2. ERM Architecture

ERM provides a framework for monitoring any finite resource within the Cisco IOS software and provides information that a user can analyze to better understand how network changes might impact system operation. ERM helps in addressing infrastructure problems such as reloads, memory allocation failure, and high CPU utilization by performing the following functions:

- Monitoring system resource usage.
- Setting the resource threshold at a granular level.
- Generating alerts when resource utilization reaches the specified level.
- Generating internal events using the Cisco IOS Embedded Event Manager feature.

A. Mechanism

ERM tracks the resource usage for each RU internally. An RU is a subsystem or process task within the Cisco IOS software; for example, the Open Shortest Path First (OSPF) hello process is a resource user. Threshold limits are used to notify network operators of specific conditions. The ERM infrastructure provides a means to notify the internal RU subsystem of threshold indications as well. The resource accounting is performed by individual ROs, resource owners. ROs are part of the Cisco IOS software and are responsible for monitoring certain resources such as the memory, CPU, and buffer. When the utilization for each RU exceeds the threshold value set, the ROs send internal notifications to the RUs and to

network administrators in the form of system logging (syslog) messages or Simple Network Management Protocol (SNMP) alerts.

We can set rising and falling values for critical, major, and minor levels of thresholds. When the resource utilization exceeds the rising threshold level for a configured period of time, an Up notification, aka Rising warning, is sent. When the resource utilization falls below the falling threshold level for a configured period of time, a Down notification, aka Falling warning, is sent.

ERM provides for three types of thresholds to be defined:

- The System Global Threshold is the point when the entire resource reaches a specified value. A notification is sent to all RUs once the threshold is exceeded.
- The User Local Threshold is the point when a specified RUs utilization exceeds the configured limit.
- The User Global Threshold is the point when the entire resource reaches a configured value. A notification is sent to the specified RU once the threshold is exceeded.

This paper tackles the critical level of the System Global Threshold applied to memory, which means we don't focus on each and every resource user's memory consumption, but rather on the memory utilization of all resource users all at once.

ERM performs the accounting of information for memory by tracking the memory usage of individual RUs. When a process is created, a corresponding RU is also created, against which the usage of memory is recorded. The process of RU creation helps the user to migrate from a process-based accounting to a resource user-based accounting scheme for memory.

ERM consists in mainly two predefined upper and lower thresholds called Rising and Falling thresholds, respectively. A Rising warning is generated meaning resource depletion may occur any time soon. Whereas falling below the Falling threshold indicates that the system is back to normal.

The following example shows how to configure a global policy with the policy name as system-global-pc1 for processor memory with critical threshold values of 90 percent as rising at an interval of 20 seconds, 60 percent as falling at an interval of 10 seconds:
See the following command :

```
$ configure terminal
$ resource policy
$ policy system-global-pc1 global
$ system
$ memory processor
$ critical rising 90 interval 20 falling\
60 interval 10
```

If a Rising warning is followed by a resource depletion, this means that the Rising warning will be labeled as a True Positive, because we were able to warn the user in advance. But if the Rising warning is followed by a Falling warning, this means that the generated warning is a false alarm and will be labeled as a False Positive.

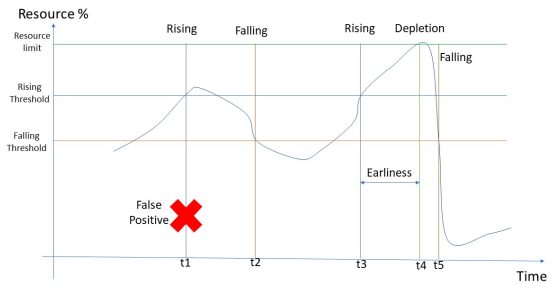

Figure 3. ERM

B. Drawbacks

ERM has several drawbacks. First, we should configure two thresholds, and in order to do this, domain knowledge is a must. Second, if the resource utilization crosses the Rising threshold, we don't know if resource depletion will ever be reached, because ERM sends warnings for every exceeding of a Rising threshold. Therefore, ERM warnings are not reliable enough, a lot of False Positives are expected from this method. And finally, if the Rising warning was followed by resource depletion, we don't know when resource depletion will take place, it could be one hour, one day or even one week after the Rising warning. ERM does not provide any information on the expected time of the upcoming depletion. This is why we decided that $Closeness$ will be equal to the value of $Earliness$, which is the difference in time between the sending of a Rising warning and resource depletion, because ERM doesn't predict when depletion will take place.

V. Forecasting-based method

In this paper, the main objective is to introduce a new approach to warning generation based on forecasting. At each and every timestamp, a set of predictions is made for the values that the monitored resource will eventually go through. This way, we know in advance when depletion will occur as opposed to the threshold-based method.

A. Mechanism

Fig.4 shows a set of predictions made at $t1$. This set shows that the resource limit will be reached at $t'1$, so a warning is sent at time $t1$ saying that the resource will be depleted at $t'1$. But in this case, it is seen that depletion happened at time $t2$. Nevertheless, this warning will still be labeled as a True Positive and the $Earliness$ will be the time difference between $t2$ and $t1$, and the $Closeness$ will be the difference between $t2$ and $t'1$. The closer the $Closeness$ to zero the better because it means that the prediction was perfectly accurate.

Instead of generating alarms when thresholds are crossed, the last monitored values are taken and used in order to

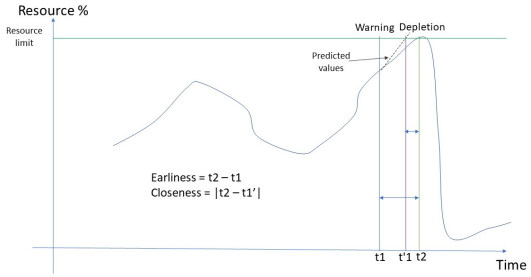

Figure 4. Forecasting-based method

predict the potential next values the monitored resource will go through, and based on that, the system decides whether to generate a warning or not.

Fig.5 illustrates the architecture of the forecasting-based method that can be decomposed into two components: the first one is the Forecaster and is responsible for predicting the next values, and the second component is the Detector and is in charge of deciding when a warning should be generated.

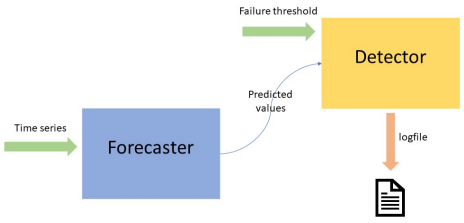

Figure 5. Forecasting-based Architecture

B. Forecaster

In the forecasting-based method, a univariate approach was taken, meaning that only one feature was taken for input, ignoring all the other time-series, just like ERM. The decision behind the univariate approach is backed up by several facts given in section VI.

The Forecaster takes the last monitored values in order to predict the following ones. Three different models were implemented for time series forecasting: The first one is the Auto-regressive method [1]. It's a linear learning model that struggles with non-linearities. This model was not only implemented for its simplicity but also because the data show a sawtooth pattern, shown in Section VI, and its tendency can be captured by the Auto-regressive mechanism. This is a common approach for classical statistical time series forecasting. Two different approaches were used for this method. The first one is the Random sample consensus approach, aka Ransac [3]. Ransac is a learning technique to estimate parameters of a model by random sampling of observed data, it uses the voting scheme to find the optimal fitting result. The second one is by using the huber loss function [5]:

$$L_\delta(y, f(x)) = \begin{cases} 0.5 * (y - f(x))^2, & |y - f(x)| \leq \delta \\ \delta * (|y - f(x)| - 0.5 * \delta), & otherwise \end{cases}$$

Next, the Feed-Forward Neural Network, Multilayer Perceptron, is a learning model that is challenged with output noise in the data. MLP was used because it is very flexible and can be used generally to learn a mapping from inputs to outputs. The results can be used as a baseline point of comparison to confirm that other models that may appear better suited add value.

And finally, the state-of-the-art Recurrent Neural Network for time series forecasting, LSTM [13], is also a learning model. Long Short-Term Memory (LSTM) is a type of recurrent neural network that can learn the order dependence between items in a sequence. LSTMs have the capability of being able to learn the context required to make predictions in time series forecasting problems, rather than having this context pre-specified and fixed. But there is some doubt as to whether LSTMs are appropriate for time series forecasting.

C. Detector

The Detector is in charge of generating alarms based on the predicted values. It also faces several challenges:

1) Given a single set of predictions (made at a given timestamp), should we generate a warning ?
2) Given a multiple set of predictions (made at successive timestamps), how/when should we generate a warning ?
3) Given a multiple set of predictions , how do we deal with redundant warnings ?

Let's combine the first two challenges into Fig.6 and forget about the third one for now. As shown in this figure, every set of predictions is connected to the timestamp where these values were predicted on their left. The blue boxes mean that the corresponding value is below the resource limit and the orange ones mean that the predicted value has exceeded the resource limit.

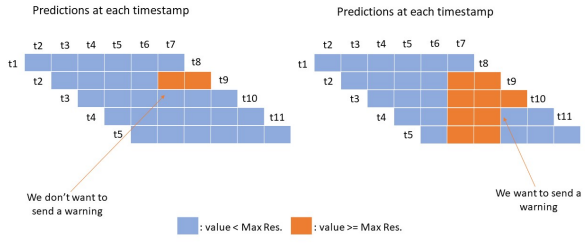

Figure 6. Generating warnings

On the left, it is seen that the set of predictions made at $t1$ doesn't have any value above the resource limit, so the

system decides to do nothing. Then at $t2$, orange boxes are seen at timestamp $t7$ and $t8$, which means that the value colored in orange has reached the resource limit. At first, we are prone to send a warning, but if we decide to wait, it is clearly shown at time $t3$ that it's not the case anymore because all the values have turned to blue. In this case, reaching depletion at $t7$ is considered to be highly improbable so the system decides to do nothing. This is one way to deal with noise and outliers inside the set of predictions. For the next example, it is seen the persistence of the orange color at time $t7$ that was predicted at several consecutive timestamps. In this case, it is considered that it's highly probable to reach depletion at $t7$, and therefore decide to generate a warning. For this purpose, two internal variables were created, one that keeps track of the time depletion was predicted to happen, and the other, named $nb\_of\_confirmation$ that represents the number of timestamps that the system should wait before sending a warning, so as to decrease the chances of sending a false alarm.

In this paper, regardless of the method used for time-series forecasting, we always predict 24 steps ahead. If at a given timestamp it is predicted that depletion will occur for the first time 23 steps ahead, then $nb\_of\_confirmation$ is set to a large value, for example 5, which means that the system waits and sees if depletion will occur in the five following timestamps before generating a warning. On the other hand, if at a given timestamp it is predicted that depletion might occur for the first time in the following 2 steps, then $nb\_of\_confirmation$ is set to 0, which means that the system should not wait for confirmation and send a warning, because the timestamp at which the resource utilization reaches its limit is really close. Also, the variable $nb\_of\_confirmation$ will be decreased only if the the predicted time of depletion made at the current timestamp is less or equal to the previous one. This way, if it's being pushed away from one timestamp to another, then $nb\_of\_confirmation$ will not be decreased and the second condition, that is essential for generating a warning, won't be satisfied.

If we take the third challenge and forget about the first two, it is seen in Fig.7. how redundant warnings are dealt with. Before going any further, we notice a third color that was added to the previous ones. This color was introduced in order to better understand the notion of intervals. First, this figure shows that no resource depletion was predicted at timestamps $t1$, $t2$ and $t3$, so the system did nothing. Then, at $t4$, depletion is predicted to happen at $t8$, so a warning is generated at $t4$ stating that the device will be out of resource at $t8$. An interval is created around $t8$, in this example, the interval width is set to 1, which means the timestamp just before resource depletion and the one just after it are included, so the interval will be equal to $[t7, t9]$. This interval will be crucial to deal with redundant warnings. Afterwards, at time $t5$, resource depletion is predicted to happen at $t7$ that is different from $t8$, so before canceling the previous warning, the system checks if the predicted time of depletion falls inside the interval, and if it's the case, it does nothing because it tolerates a small margin of error. So the bigger the interval width, the bigger the margin of error. Same thing happens at $t6$ where the predicted time of resource depletion $t9$ is contained inside $[t7, t9]$. But then, at $t7$, it is viewed that the resource limit is attained at $t12$, and $t12$ is outside $[t7, t9]$. So in this case, the first warning is canceled and a new one is generated by updating the predicted time of resource exhaustion and set a new interval around $t12$ equal to $[t11, t13]$.

In some cases, we may want to cancel a warning that was previously sent without creating a new one. This happens when there is a sudden change in the data, huge chunks of the monitored resource were freed for example, and resource utilization is no longer near the resource cap value. In this case, a new variable is introduce: $end\_of\_depletion$, so that whenever a warning is sent, if the resource utilization falls below the value held by $end\_of\_depletion$, the previously sent warning gets canceled.

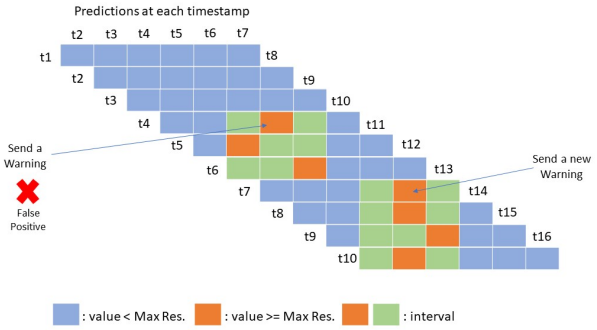

Figure 7. Dealing with redundant warnings

The detector's architecture is shown in Fig.8 after combining all three challenges.

the system starts by checking if the latest monitored value is greater or equal to the resource limit value. If it's the case, then it does nothing because it's too late to generate a warning. Elsewhere, it faces the first challenge which is checking the set of predictions and checks for the presence of a value greater or equal to the resource limit.

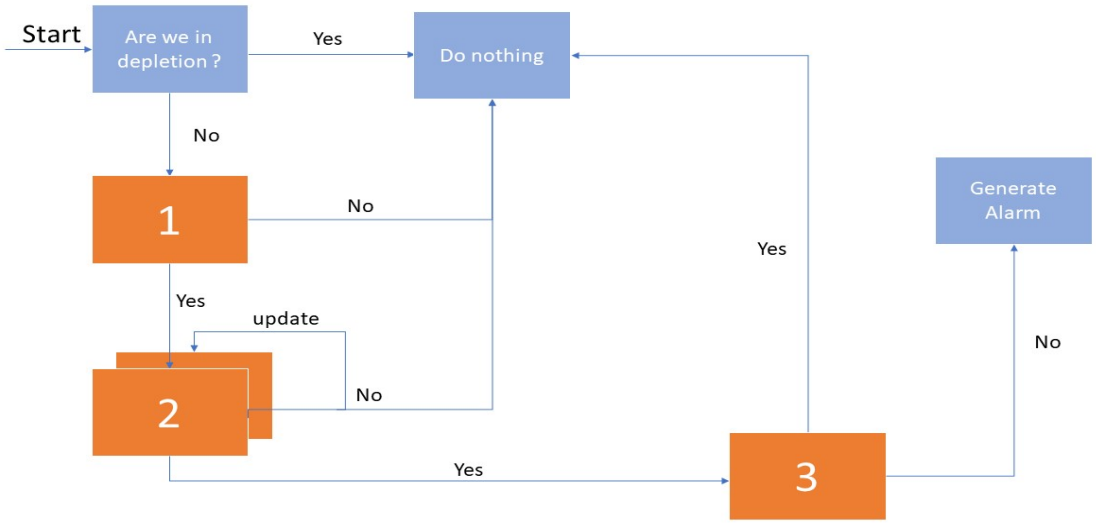

Figure 8. Detector's architecture

Algorithm 1: Challenge 1

```
/* initially, nb_of_confirmation = -1, when this
   value is equal to zero, a warning should be
   generated.                                    */
/* predictions is the set of predictions made at
   the current timestamp.                        */
/* t_now is the current timestamp.               */
/* t_depletion is the predicted time of depletion
   made at t_now.                                */
/* depletion is a boolean value that is set to
   True if a value in predictions reaches the
   limit.                                        */
initialization;
for t in range(len(predictions)) do
    if predictions[t] > resource_limit then
        t_depletion = t_now + t + 1;
        depletion = True;
        break;
if not depletion then
    nb_of_confirmation = -1;
    return;
```

If no value is greater or equal to the resource cap in the predictions, nothing is done because we don't fear of reaching the limit any time soon. But if it's the case, then the system moves on to the second challenge.

Challenge 2 Algorithm introduces the notion of confirmation. Before proceeding on to the next challenge, the $nb\_of\_confirmation$ variable must be set to 0.

Finally, the third and last challenge is described in Challenge 3 Algorithm. If the predicted time of depletion is inside the interval, then nothing is done because we don't want to generate redundant warnings. Elsewhere, the earlier sent warning is canceled before generating a new one and updating the interval surrounding the predicted time of depletion.

The Detector can be seen as a finite-state machine where two final states can be reached, hence two actions can be taken: It either does nothing or generates a new warning. The Detector's architecture can be seen in Fig.8.

Once all the conditions satisfied, the Detector sends a warning by logging it to a pre-specified logfile. The logfile will contain the warnings in the following format:

Algorithm 2: Challenge 2

```
/* t_last_depletion is the predicted time of
   depletion made at the previous timestamp. */
/* warning is a boolean that is set to True if
   the answer to the second challenge is yes.
   */
initialization;
warning = False;
dist_now = (t_depletion - t_now);
dist_predicted = (t_depletion - t_last_depletion);
if nb_of_confirmation < 0 then
    nb_of_confirmation = dist_now // 4;
if nb_of_confirmation == 0 then
    warning = True;
    nb_of_confirmation = -1;
else if dist_predicted <= 0 then
    nb_of_confirmation -= 1;
else
    nb_of_confirmation = dist_now // 4;
```

Algorithm 3: Challenge 3

```
/* interval is a tuple containing the lower and
   upper bound of the interval seen in the
   previous section.                              */
/* interval_size is an integer used for computing
   the interval.                                  */
initialization;
if warning then
    if t_depletion < interval[0] or t_depletion >
     interval[1] then
        t_last_depletion = t_depletion;
        interval[0] = t_depletion - interval_size;
        interval[1] = t_depletion + interval_size;
        /* generate new warning                   */
        ...
```

```
WARNING:root:t+3068, The resource
will reach 90.036684 at t+3084.
WARNING:root:Falling: t+3080.
WARNING:root:t+4539, The resource
will reach 90.004516 at t+4554.
WARNING:root:Falling: t+4551.
WARNING:root:t+5256, The resource
will reach 90.017265 at t+5271.
WARNING:root:Falling: t+5273.
```

VI. Experimental setup

In this section, we describe the data that was used for testing and the experimental setup.

A. Lab Topology

The lab was design and built around the same methods and mindset as the web/hyperscale customers use for their data centers and routed backbones. This carries in to the cabling, IP and BGP addressing schemes for operational efficiency, programability and automation [12]. Fig.9 shows the lab topology.

B. Model-driven Telemetry

The resource used for testing was the free application memory. It was gathered from devices in our lab, Cisco NCS 55A1, running IOS-XR [16] with Model-Driven Telemetry [10].

Model-driven telemetry is a new approach for network monitoring in which data is streamed from network devices continuously using a push model and provides near real-time access to operational statistics. Applications can subscribe to specific data items they need, by using standard-based YANG data models over NETCONF-YANG. Cisco IOS XE streaming telemetry allows to push data off of the device to an external collector at a much higher frequency, more efficiently, as well as data on-change streaming [10].

We were able to gather more than 7k time-series, aka features, throughout 5 months with a 10-second cadence between the data points. The data was gathered from the following devices: leaf1->8, spine1 and spine2. For each and every device, we took the memory utilization of node 0 and node RP0, therefore extracting two datasets from one single network device.

For the threshold-based method, only one feature was used and it is the free application memory. From this feature, we are able to derive the memory resource utilization that will be piped to ERM's input in order to generate Rising and Falling warnings.

As for the forecasting-based method, a univariate approach was taken, meaning that only one feature was taken for input, ignoring all the other time-series, just like ERM. The decision behind the univariate approach is backed up by several facts. First, feature selection algorithms were applied on the set of time-series in order to pick the best features that can improve the predictability of the free application memory. After removing all the features that had constant values, we were left with approximately 1500 features. We used the remaining features as input for Boruta [6]. Boruta is a popular algorithm for feature selection. It's capable of capturing non-linear relationships and interactions. After fitting the model with the data, we extract the importance of each feature and keep only the ones that are above a given threshold of importance. We have derived the rank of the most important features for the target $Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary[node-name:0/RP0/CPU0]free-application-memory$ and this was the result:

- Feature: $Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/detail[node-name:0/RP0/CPU0]free-application-memory$ Importance: 0.9195635315030917
- Feature: $Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary[node-name:0/RP0/CPU0]free-physical-memory$ Importance: 0.018317492024895507
- Feature: $Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/detail[node-name:0/RP0/CPU0]total-used$ Importance: 0.010372193501278926
- Feature: $Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/detail[node-name:0/RP0/CPU0]free-physical-memory$ Importance: 0.009107382363270434

Put differently, only one feature is considered important for the prediction of free application memory, and this feature is itself.

Also, by taking a univariate approach, we consume less resources and require less computation power as opposed to the mutlivariate approach, specially if the aim is to run the forecasting-based method onbox.

C. Data visualization

After selecting the free application memory, deriving the total memory utilization and plotting it, it is seen in Fig.10

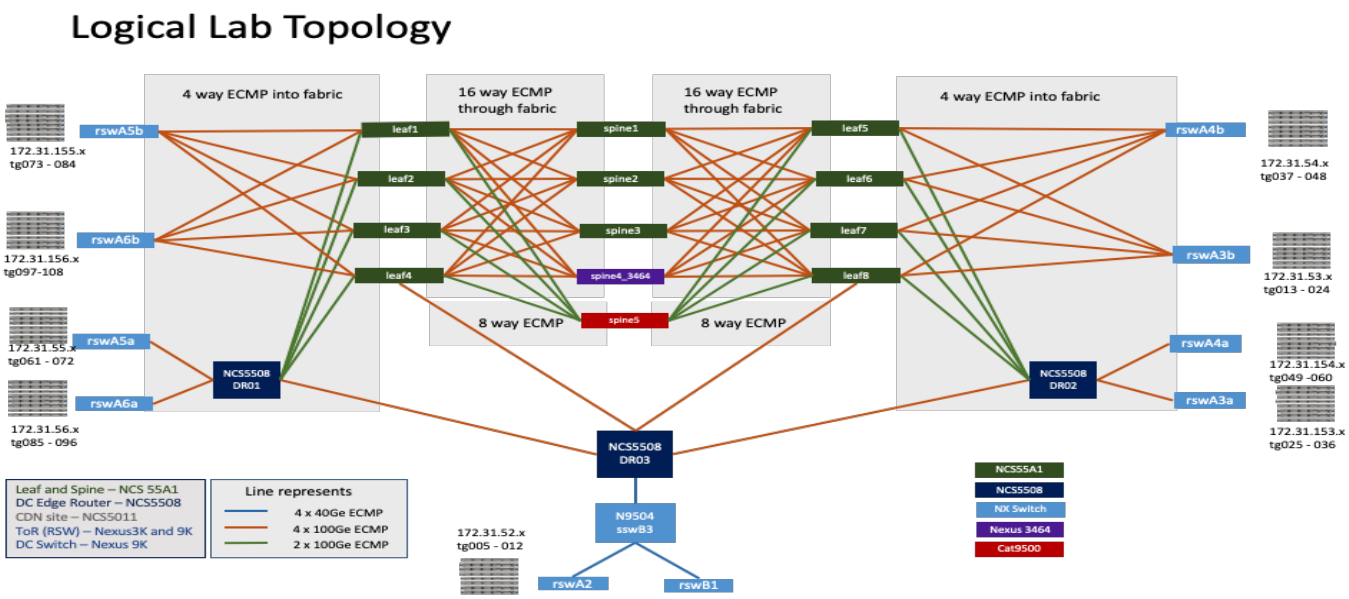

Figure 9. Lab topology

and Fig.11 that memory utilization in our lab shows a sawtooth pattern that is caused by a memory-leak bug.

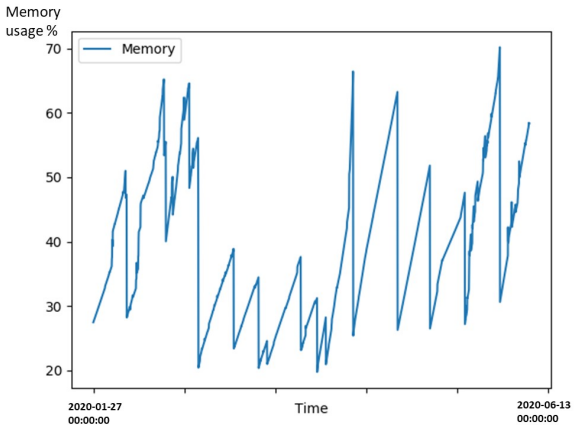

Figure 10. Memory utilization in leaf5 node RP0

We decided to benchmark Memory's behavior using ERM and the forecasting-based methods in order to compare their performances.

The data was transformed from a 10-second to a 10-minute cadence so that we would be able to predict farther into the future, because our forecasting-based methods predict 24 time-steps ahead. The bigger the value of the forecasting horizon, the more computation power is required and the less accurate we get. Therefore, with a 10-second cadence, we are able to predict at most 4 minutes in advance whereas we are able to predict at most 4 hours in advance with the 10-minute cadence.

D. Experimental Setup

All of the methods and algorithms were implemented in python3.

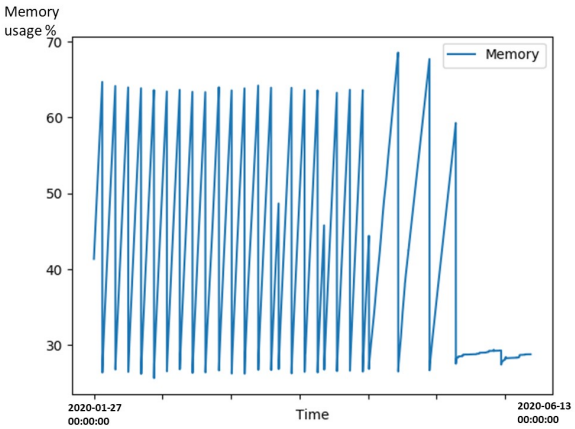

Figure 11. Memory utilization in leaf1 node 0

1) ERM: ERM is a Cisco tool that was developed and used in IOS-XE and isn't available in IOS-XR. So the implementation of this tool was done in python. It takes as input a time-series of monitored values (resource values like CPU, Memory and buffers), two thresholds, the rising and the falling, and with their respective time interval thresholds, for every resource selected, and returns a notification if a specific resource exceeded the threshold for more than the specified time interval. The time-series values are expected to be represented in percentage values (0% to 100%).

2) AR: The Auto-regressive approach is a forecasting-based method that was implemented with scikit-learn [11]. Several Auto-regressive methods can be found inside the $linear\_model$ library. After implementing 10 linear models and comparing their performances, we decided to keep only 2 out of 10: The Ransac Regressor and the Huber Regressor.

Algorithm 4: Regressors

```
def get_models(models=dict()):
    models['lr'] = LinearRegression();
    models['lasso'] = Lasso();
    models['ridge'] = Ridge();
    models['en'] = ElasticNet();
    models['huber'] = HuberRegressor();
    models['lars'] = Lars();
    models['llars'] = LassoLars
    models['pa'] =
     PassiveAggressiveRegressor(max_iter=1000,
     tol=1e-3);
    models['ransac'] = RANSACRegressor();
    models['sgd'] = SGDRegressor(max_iter=1000,
     tol=1e-3);
    return models;
```

3) MLP: The MLP is used as a point of reference for all other modeling techniques. In order to choose the best parameters for the MLP, we performed hyper-parameters tuning with Talos [15], and chose the model with the smallest validation loss value. The parameters that were studied are :

```
p = {'activation':['relu', 'elu'],
     'hidden_layers':[1, 2],
     'dropout': [0, 0.05, 0.1],
     'batch_size': [16, 32, 64],
     'epochs': [100],
     'input_window':[24],
     'output_window': [24],
     'neurons_1': [2,4,6,8],
     'neurons_2': [2,4,6,8]}
```

With $dropout$ the value of the dropout layers situated right after the hidden layers, $neurons\_1$ the number of perceptrons for the first hidden layer and $neurons\_2$ the number of perceptrons for the second hidden layer in case $hidden\_layers$ is equal to 2.

After running the script, the model that was retained had a validation loss of 0.4579317, and can be seen in Fig.12.

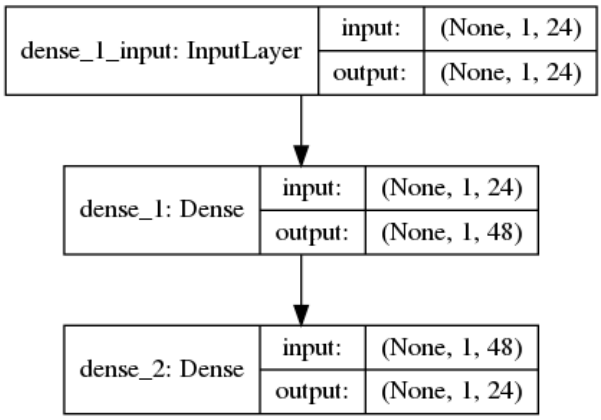

Figure 12. Multi-layer perceptron

4) LSTM: LSTM is a Recurrent Neural Networks (RNNs) that can be used for Time Series Forecasting. They are designed for Sequence Prediction problems and time-series forecasting nicely fits into the same class of problems. We performed hyper-parameters tuning with Talos, and chose the model with the smallest validation loss value. The parameters that were studied are :

```
p = {'activation_lstm':['relu', 'elu',
    'softmax'],
    'activation_dense':['relu', 'elu',
    'softmax'],
     'add_hidden_dense':[0, 1],
     'dropout': [0, 0.05, 0.1],
     'batch_size': [16, 32, 64],
     'epochs': [100],
     'input_window':[24],
     'output_window': [24],
     'neurons_1': [2,4,6,8],
     'neurons_2': [2,4,6,8]}
```

With $neurons\_1$ the number of cells for the first hidden LSTM layer and $neurons\_2$ the number of cells for the second hidden LSTM layer, and add_hidden_dense whether a Dense layer should be added after the second hidden LSTM layer or not.

After running the script, the model that was retained had a validation loss of 0.4566888, and can be seen in Fig.13.

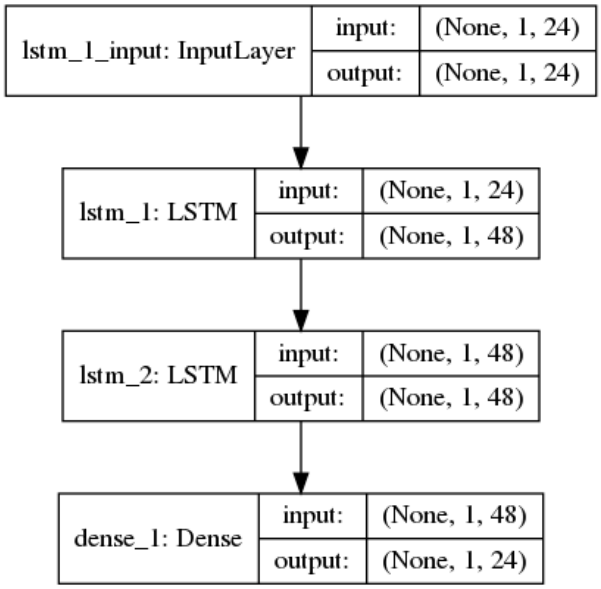

Figure 13. Long short-term memory

In the next section, the resource limit was set to 61% since most of our datasets don't exceed the value of 65%. This way, by setting the resource limit to a lower value, we will be able to gather a larger set of ground-truth with plenty events marked as depletion, 165 to be precise. We don't consider setting the resource cap to a lower value as misleading nor degrading the quality of the warning generators, because if we are able to

successfully warn the user in advance before we reach 61%, then we will also succeed at doing so with a larger and more realistic value.

VII. Results

As mentioned earlier in Section III, several aspects has been introduced. In the following, the aspects will be used by means of metric so that comparison between different methods and approaches will be possible. The formulae for computing the metrics are given below:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$Earliness = \frac{1}{TP} \sum_{n} t_{depletion}(n) - t_{warning}(n)$$

$$Closeness = \frac{1}{TP} \sum_{n} |t_{depletion}(n) - t_{predicted\_depletion}(n)|$$

A. ERM trade-off Precision/Earliness

The benchmark results for ERM can be seen in Fig.14 and table.I. The data fed to ERM was downsampled to 10-minute cadence. In practice, ERM works on data with small steps in time between two data points, hence the use of the time intervals for the thresholds. But in this case, the data was pre-processed before being fed to any of the models, whether it is threshold-based or forecasting-based. For this reason, the time intervals will be ignored, the moment the Rising threshold is crossed, a new warning is generated.

The results were obtained after testing ERM with a Rising threshold varying from 50% up to 60%, with a Falling threshold of 45% on all 20 datasets gathered from 10 devices as specified in the previous section.

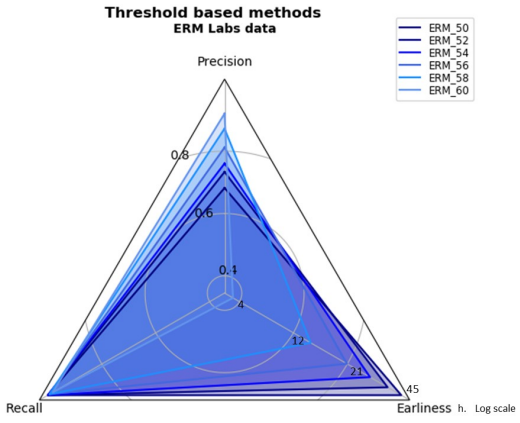

Figure 14. ERM's trade-off

Table I
ERM results

$RisingTh.$	$Precision$	$Recall$	$Earliness(hours)$
50	0.68	1.0	44.66
52	0.73	1.0	36.85
54	0.76	1.0	28.77
56	0.81	0.994	20.78
58	0.87	0.994	12.47
60	0.92	0.982	4.17

These results shows a clear trade-off between the $Earliness$ and $Precision$ present in ERM. It can be explained by the fact that the closer the Rising threshold to the resource limit, the more precise the system gets, because the number of False positives is reduced. For example, we are more likely to reach depletion if resource utilization crosses the 60% threshold than the 50% threshold. As for the $Earliness$, the lower the Rising threshold bar the sooner we get notified.

In the following, ERM with a Rising threshold of 60% was picked for comparison with our proposed solution considering that $Precision$, $Recall$ and $Closeness$ are more important than $Earliness$, as long as the value of the latter is reasonable, greater than 1 hour so appropriate countermeasures can still be taken.

A small degradation is seen in the $Recall$, but this is due to missing data points in our dataset when down-sampling was applied. Initially, we had huge datasets with a 10-second cadence representing the memory utilization on a network device over 5 months. Down-sampling was applied by picking one data point and ignoring the next 59 data points before picking the 60th (10-second -> 10-minute cadence). But due to missing data in some of the original datasets, down-sampling emphasized the gap between data points and therefore obtained 3 False Negatives for ERM with a Rising threshold of 60%.

B. Threshold-based vs Forecasting-based method

The comparison between the threshold-based method represented by ERM and the forecasting-based method represented by four different models is shown in Fig.15 and table.II. In Fig.15, a new axis was added to the plot, displaying $\frac{1}{Closeness}$ score for the methods used. We decided to take $\frac{1}{Closeness}$ and not $Closeness$ itself because we want to minimize the value of the latter, and therefore by inverting it, we will have to maximize the value around all of the axis.

In view of the fact that ERM doesn't provide information related to the time of depletion, we decided to set the value of $Closeness$ equal to $Earliness$ so that ERM can be included in our comparison.

As mentioned earlier for ERM, the $Recall$ for the forecasting-based methods is also challenged by missing data points. After investigating the 4 False Negatives obtained by AR ransac, it turns out that 3 out of 4 are due to missing data points, these 3 False Negatives being also the same for ERM 60% as discussed above.

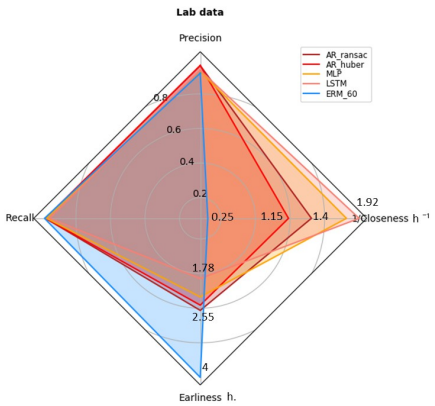

Figure 15. warning generators comparison

Table II
WARNING GENERATORS RESULTS

Method	*Precision*	*Recall*	*Earliness*	*Closeness*
AR ransac	0.964	0.976	2.55	0.71
AR huber	0.964	0.976	2.43	0.87
MLP	0.93	0.97	2.23	0.58
LSTM	0.934	0.951	1.79	0.52
ERM 60	0.92	0.982	4.17	4.17

In order to pick the best model for warning generation based on the results, we proceed as follows by introducing two new scores:

$$F1score = 2.\frac{Precision\ .\ Recall}{Precision + Recall}$$

$$ECratio = \frac{Earliness}{Closeness}$$

The $F1score$ will be prioritized to $ECratio$ due to the fact that all methods have an $Earliness$ greater than 1 hour. After selecting the best models based on their F1-score, the last selection will be based on the ratio between $Earliness$ and $Closeness$.

Table III
WARNING GENERATORS RESULTS

Method	*F1score*	*ECratio*
AR ransac	**0.97**	**3.59**
AR huber	0.97	2.79
MLP	0.95	3.84
LSTM	0.942	3.44
ERM 60	0.95	1.0

Based on the results presented in table.III, the methods with the highest $F1score$ are AR ransac and AR huber, and based on the $ECratio$, we prove that AR ransac is the best solution for our problem.

Finally, The fifth and final aspect to be taken into consideration is the resource consumption for each method shown in Fig.16. It is seen that ERM is light weight and this is due to the fact that it is a naive simple threshold-based method. As for the forecasting-based methods, none of the models used consume more than 600 kB of ram memory, which is very little. And for the computational power, it is seen that LSTM takes a lot of time to train as opposed to the others.

The results below show that the Auto-regressive approach and MLP can be run on-box due to their light resource consumption.

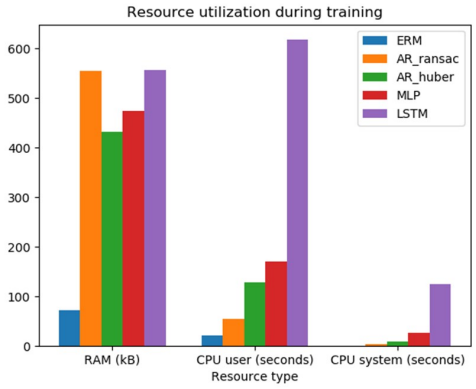

Figure 16. Resource consumption per method

VIII. DISCUSSION AND FUTURE WORK

There are many directions of future work discussed in this section.

A. data generator

The data available in our lab shows specific traits and its access cannot be made public, hence the creation of a data generator. The latter is an empirical model where parameters can be tuned to get different scenarios not available in our lab.

With the data generator, we are able to make a completely different case from the lab's cases. For example, the number of processes is stable in the lab. Routers run no/few user process that can be initiated at will, and in general, only one process is leaking. In order to emulate a network device and collect the memory utilization of all the running processes, we have three configurable proceedings.

The first one is the birth of new process, it can be explained with the poisson distribution, $P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$, for the purpose of emulating it (no longer stable process count, overall process number has different dynamism).

A skewed distribution can be used for life expectation. Some processes run for a long period of time, like system processes, and some die faster, like user processes, not daemons.

Finally, the normal distribution is used for individual memory allocation. We can control where at individual process level there is leak or not.

The purpose behind the data generator is to have data that have different features, yet through parameters setting, we can make it similar to lab data. And in order to prove this, we generated datasets with the data generator where the resource utilization was able to reach 90% (The data generator was designed to kill the top 3 processes consuming memory whenever the total memory utilization reaches 90% by mimicking the behavior of Resmon in IOS-XR). In the latter case, the resource limit was set to 90%.

The Fig.17 shows the plot of the memory utilization that was emulated with the data generator. The same leaky behavior can be seen in this figure.

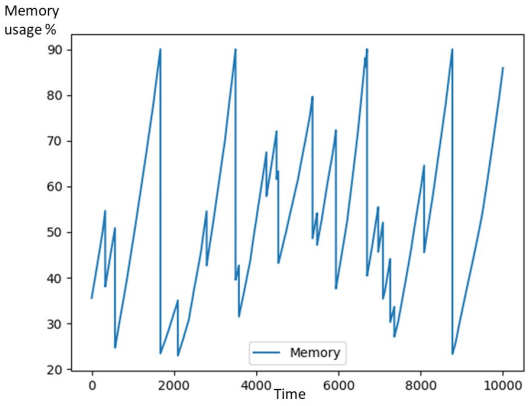

Figure 17. Generated data: Memory utilization

B. Forecaster algorithms

Three different 'Forecasters' were implemented and tested, and showed great performance compared to ERM. However, the main focus in this paper is not attributed to the Forecaster component but rather to the handling of the set of predictions, post-forecasting. Nevertheless, the algorithms implemented are very basic ones and can be replaced by high-end algorithms.

For instance, The Makridakis Competitions are a series of open competitions intended to evaluate and compare the accuracy of different forecasting methods. The M4 Competition follows on from the three previous M competitions, the purpose of which was to learn from empirical evidence both how to improve the forecasting accuracy and how such learning could be used to advance the theory and practice of forecasting [9].

The implementation of some of the methods used in the M4 competition was done using the sktime [7] library in python. Testing was not performed due to functions not implemented yet in this library.

Another way to improve the Forecaster is by changing the forecasting horizon. In this report, the forecasting horizon was fixed from $t1$ -> $t24$, meaning the Forecaster is predicting 24 consecutive steps ahead. Instead of predicting consecutive steps, the Forecaster can predict several steps ahead with gaps between the timestamps. For example, if the current timestamp is $t0$, predict the values for timestamps: $t1$, $t8$, $t15$, $t22$, and $t29$, thus decreasing the output size, consequently the number of weights used by the model, while stretching our predictions into the future.

C. Per PID analysis

The warning generators presented in this paper took a univariate time-series as input with the intention of generating warnings before resource depletion. Another procedure was taken where instead of feeding the warning generator one single value that represents the total memory consumption, a whole set of values where each Process ID is associated with its own memory utilization is fed. Consequently, the system will be able to generate more meaningful warnings by showcasing the 3 processes with the highest memory consumption, and the 3 processes with the highest deltas with regard to memory utilization, identifying the leaky processes. This approach was implemented in linux, where the memory utilization was continuously monitored and its output piped to the warning generator.

D. Detector's error back-propagation

The Detector is a finite-state machine. This component can be extended to a learning component where the errors can be back-propagated so that its parameters get auto-configured. We believe that introducing new parameters to the Detector, like False Positive count, False Negative count, etc., can be used to find out if the parameters and weights of the model in use need to be updated, and whether the Forecaster needs retraining or not. As a matter of fact, the monitored resource might change its behaviour over time. If the Forecaster is not retrained on the newly received data, then it will hardly be able to catch the features of the monitored resource, thus giving inaccurate set of predictions to the Detector.

E. Potential IOS-XR integration

We were given the opportunity to present our work to the IOS-XR team where they found the idea very interesting. We already had a meeting where we walked them through the code and explained the mechanism behind the forecasting-based method. Some of the questions raised were introduced earlier in this section as future work possibilities.

IX. Conclusion

In this paper, a warning generator was developed based on forecasting, an enhanced machine learning based framework for predicting the usage of various resources in network devices in the interest of sending early warnings to resource depletion.

The forecasting-based method presented in this paper takes a univariate time-series as input representing the monitored resource, a machine learning algorithm used for time-series forecasting, and a finite state machine to decide whether a warning should be generated or not.

We extensively evaluate our solution on predicting memory usage on a set of 10 devices over a period of 5 months and

compare its performance to the threshold-based method used in production, Embedded Resource Manager.

We are able to achieve a better performance in terms of $Precision$ by following a predetermined set of conditions before sending the warnings, thus reducing the number of false alarms. Also, ERM does not provide any information related to the time of depletion, whereas our approach provides such information with an average error of 42 minutes if the Autoregressive ransac approach is adopted as our prediction model, and therefore bringing a novelty to this field of study.

Thanks to the forecasting-based approach, we are able to efficiently generate meaningful warnings with information related to the date of depletion, the processes with the highest memory consumption and the ones with the highest memory deltas, in contrast to the deployed model in production.

For our future work, we intend to develop new methods for the Forecaster component and transform the Detector to a learning component that is able to back-propagate the errors and retrain the model in use whenever needed.

Acknowledgement

The authors would like to thank Parisa Foroughi, Anil Kuriakose and Shwetha Bhandari for their suggestions and valuable discussion.

References

[1] Hirotugu Akaike. "Fitting autoregressive models for prediction". In: *Annals of the Institute of Statistical Mathematics* 21 (1969), pp. 243–247. ISSN: 1572-9052. DOI: 10.1007/BF02532251. URL: https://doi.org/10.1007/BF02532251.

[2] *Embedded Resource Manager Configuration Guide, Cisco IOS Release 15MT*. 2012. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/erm/configuration/15-mt/erm-15-mt-book.pdf.

[3] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[4] N. Gutierrez and M. Wiesinger-Widi. "AUGURY: A Time Series Based Application for the Analysis and Forecasting of System and Network Performance Metrics". In: *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. 2016, pp. 351–358.

[5] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *Ann. Math. Statist.* 35.1 (Mar. 1964), pp. 73–101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[6] Miron Kursa, Aleksander Jankowski, and Witold Rudnicki. "Boruta - A System for Feature Selection". In: *Fundam. Inform.* 101 (Jan. 2010), pp. 271–285. DOI: 10.3233/FI-2010-288.

[7] Markus Löning et al. "sktime: A Unified Interface for Machine Learning with Time Series". In: *Workshop on Systems for ML at NeurIPS 2019*. 2019.

[8] *Maintaining System Memory Configuration Guide, Cisco IOS Release 15MT*. 2018. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sys-mem-mgmt/configuration/15-mt/sysmemmgmt-15-mt-book/sysmemmgmt-mem-leak.html.

[9] Spyros Makridakis, Evangelos Spiliotis, and Vassilis Assimakopoulos. "The M4 Competition: Results, findings, conclusion and way forward". In: *International Journal of Forecasting* (June 2018). DOI: 10.1016/j.ijforecast.2018.06.001.

[10] *Model-Driven Telemetry*. URL: https://www.cisco.com/c/en/us/solutions/service-provider/cloud-scale-networking-solutions/model-driven-telemetry.html.

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] Drew Pletcher. *Lab Topology*. 2012. URL: https://wiki.cisco.com/display/WADJET/Lab+Topology.

[13] Sepp Hochreiter Jürgen Schmidhuber. "Long Short-Term Memory". In: *Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence* 385 (2012), pp. 37–38. DOI: 10.1007/978-3-642-24797-2_4. URL: https://doi.org/10.1007/978-3-642-24797-2_4.

[14] V. Sor et al. "Improving Statistical Approach for Memory Leak Detection Using Machine Learning". In: *2013 IEEE International Conference on Software Maintenance*. 2013, pp. 544–547.

[15] Autonomio Talos. *Talos*. https://github.com/autonomio/talos. 2020.

[16] *Telemetry Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 6.1.x*. 2019. URL: https://www.cisco.com/c/en/us/td/docs/iosxr/ncs5500/telemetry/b-telemetry-cg-ncs5500-61x/b-telemetry-cg-ncs5500-61x_chapter_010.html.

[17] *Using Watchdog to Always Keep a Machine Running*. May 2019. URL: https://www.supertechcrew.com/watchdog-keeping-system-always-running/.

[18] Wei Xu et al. "Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers". In: *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*. 2006, pp. 115–126.

[19] J. Xue et al. "PRACTISE: Robust prediction of data center time series". In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 126–134.