A Machine Learning-Based Framework to Estimate the Lifetime of Network Line Cards

Juan Luis Herrera*, Marco Polverini† and Jaime Galán-Jiménez*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain. [jlherrerag, jaime]@unex.es
†DIET Department, University of Rome "Sapienza", Rome, Italy. marco.polverini@uniroma1.it

***Abstract*—With the increasing tendency on data rates in forthcoming communication networks, availability is a crucial aspect to guarantee Quality of Service (QoS) requirements. The possibility of predicting the lifetime of networking hardware can be a key to improve the overall network QoS. This paper proposes a generic Machine Learning (ML) based framework that learns how to mimic the mathematical model behind the lifetime of network line cards. Results show that a good precision (85%) and recall (close to 100%) on the estimation can be achieved regardless the type of line cards the network is composed of.**

***Index Terms*—Machine Learning, line card, lifetime, estimation, QoS.**

I. Introduction

With the ever increase of the data rate in communication networks, a few milliseconds of service disruption in the data plane turn into a severe packet loss, raising problems to the Quality of Service (QoS) guarantee. In computer networks, one of the main causes of service disruption is hardware failure [1], [2].

Current solutions to recover from a network hardware failure are based on two different mechanisms: i) a failure detection method, and ii) a re-routing strategy. The main idea is to quickly detect the failure and to change the path of a subset of traffic flows, so that the failed device is bypassed. Failure detection is generally based on a monitoring system [3], [4], which sends probe packets along cyclic paths that allow to determine which one is the link that has failed. The detection of a link failure triggers the re-routing mechanism, which forces the traffic flows to avoid the failed link. For instance, recently proposed Segment Routing allows to re-route the traffic flows involved in a link failure in 50 milliseconds [5].

Considering a network Line Card (LC), its functioning can be described by the following three statuses: i) *working properly*, in which it behaves normally and loses packets only due to temporary congestion, ii) *failing*, in which it experiences temporary outages that represent another source of packet loss, iii) *died*, in which it stops working and all packets are lost. Generally, a dying LC passes from status 1 to 3, staying for a transitory time (the duration of the transitory time depends on the type of failure experienced by the LC, and can be negligible (on-off behaviour), i.e. hardware breakdown; or extended, i.e. hardware degradation) into status 2. Classical failure detection methods aim to determine if a LC is in status 3. This evaluation is done only after the hardware has died. Clearly, when the duration of the transitory time is not negligible, classical failure detection methods take a long time before activating the re-routing mechanism, leading to a consistent packet loss and QoS degradation.

In this paper, we present a Machine Learning (ML) based algorithm whose aim is the prediction of the lifetime of network hardware [6]. The prediction is then used as the triggering event for the activation of a fast re-routing mechanism. In this way, it is possible to change the traffic paths before the hardware completely stops working, highly reducing (or completely avoiding) the packet loss, and thus improving the QoS.

The proposed framework, named Line card Failure Predictor (LFP) exploits the availability of a data set (supervised learning) and an approach derived from a linear regression ML algorithm [7], to learn how to mimic the mathematical model behind the lifetime of network LCs, thus learning to estimate their remaining lifetime. Specifically, this is done by selecting, from an input set, the function shape and coefficients that are the best fit for the data set, according to a Mean Square Error (MSE) metric.

Due to the lack of real data sets on the lifetime of LCs, a network fault simulator based on mathematical models is presented to generate the input data set for the proposed LFP algorithm. We stress here that the use of synthetic data to train and test the performance of LFP does not affect the generality of the proposed approach.

To summarize, the main contributions of this work are:

- Proposal of the LFP framework for the estimation of the lifetime of network line cards.
- Creation of an extensible network failure simulator based on mathematical models.
- Training and performance evaluation of different ML systems to solve the problem at hand.

The rest of the paper is organized as follows: in Sec. II we describe the proposed system to predict the remaining life time of network line cards. Sec. III contains the performance evaluation, while conclusion and future works are presented in Sec. IV.

II. Machine Learning based Line-Card Lifetime estimation Framework

The proposed framework is intended to be integrated in a centralized monitoring system (Fig. 1). Firstly, the LFP

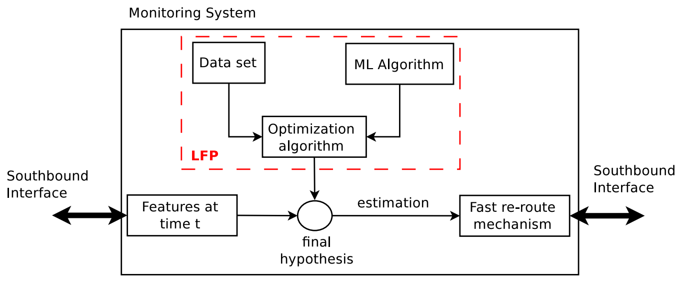

Figure 1. Architecture of the Monitoring System based on LFP algorithm.

algorithm is used to generate a final hypothesis, i.e., a function that mimics the mathematical model that regulates the lifetime of network line cards. Then, the final hypothesis is used to predict the current lifetime, by providing it with the current set of features (which will be described later) as input. Features are periodically collected from the network devices by means of a southbound interface, which can be realized by different protocols (OpenFlow, SNMP, etc.). No further functions need to be implemented, since simple OpenFlow OFP.STAT or equivalent messages could be used to get the required data, e.g., TX/RX traffic counters. Finally, the output estimation is used to eventually trigger the activation of the fast re-routing mechanism. In the next subsections, the main functional blocks reported in Fig. 1 are detailed.

A. Data Set

The *data set* is a collection of data gathered by the network devices by means of the southbound interface. Generally, monitoring systems allow the collection of statistics from the network devices with a given time granularity (e.g., 1 minute in case of Netflow [8]). This time interval between consecutive data collection processes is referred to as Time Slot (TS). Each data can be seen as a point $(x_1, ..., x_n, y) \in \mathbb{R}^{n+1}\ \forall n \geq 1$ in a multidimensional space, in which y is called *output* or *expected output*, and $(x_1, ..., x_n)$ represent the *features*.

The set of features to use must i) have an impact on the lifetime of line cards, and ii) be able to be measured by the LCs. Four features have been chosen for the framework: line card utilization in current and previous TS, and packet loss percentage in current and previous TS. Adding features that use data from previous TSs allows the system to work differently when measurements are stabilized, increasing or decreasing over two consecutive TSs.

B. Machine Learning algorithm

The main goal of the proposed framework is therefore to predict how long it will take until a failing line card finally dies. Since the remaining lifetime of a network line card is a continuous value, the problem we aim to solve falls into the regression problem archetype [9]. This type of problem is very similar to linear regression problems: given a set of points, the algorithm must find the coefficient for each feature that best fits on those points, returning a function known as *final hypothesis*.

Additionally, the system uses different non-linear models (predictors or hypothesis set) such as logarithmic, sinusoidal, polynomial, etc., that add extra features to make the mathematical function predicted more flexible than a classic linear regression model [9]. However, a new problem emerges, since we must determine which function is the best fit for the data. This is done by using a technique called *validation* [9], which consists on training the system with an $80-90\%$ of the data, and then using the remaining data (called *validation set* or *cross-validation set*) to get an estimate of its generalization accuracy. This is, the similarity between the output of the system and real data. More in detail, LFP uses a technique named *10-fold validation* [10]. Instead of dividing the data set into a large training set and a smaller validation set, 10-fold validation divides the data set into 10 subsets of same size. The system is then trained 10 times, using 9 of the subsets as the training set and the remaining subset as the validation set, making sure each of the subsets is used once for validation. After this process, 10 different estimates are obtained. Thus, a combination of the data (such as the average or median) provides a reliable and stable estimate of its generalization that is not affected by randomness.

C. Optimization algorithm

While the ML algorithm described in Sec. II-B provides a model that allows the system to learn, it is the optimization algorithm that allows it to actually learn.

The optimization algorithm requires a cost function, $J(\theta)$, which estimates the difference between the values predicted by the system and the expected output [9]. Eq. 1 defines the cost function, assessed as the half of the Mean Squared Error (MSE):

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \qquad (1)$$

In eq. 1, m represents the number of training examples; $h_\theta(x)$ is the output of the ML system given the input x (also known as *value of the hypothesis function on* x); $x^{(i)}$ is a vector with the values for the features for the i-*th* training example and $y^{(i)}$ is the expected output for the i-*th* training example.

During the learning phase, the system tries to minimize $J(\theta)$ by using the training set to assess the values. Since many different optimization algorithms are able to minimize it, a modification of the L-BFGS [11] algorithm has been used.

III. Performance Evaluation

Although initial efforts have been made to provide the research community with real data about real traffic patterns [12] and VNF performance over SDN [13] to feed ML solutions, to the best of our knowledge, no public data sets related to networking hardware error metrics are available yet. In order to evaluate our generic LFP framework, a Network Failure Simulator (NFS) has been created to produce data related to LC failure. In the following, we first describe the proposed NFS, then the description of the data set used for simulations is detailed, and a performance evaluation is finally carried out to show the effectiveness of the proposed framework.

A. NFS Description

Five input parameters are required by NFS: i) a network graph: G; ii) a set of TMs, one per TS: TM_t; iii) a hardware deterioration function: f_i; iv) a maximum lifetime span: δ_t; and a weighting parameter: α. NFS is executed once per TS t by feeding it with the corresponding TM_t and the failure characterization parameters, f_i, δ_t, α, which are used to reproduce variability in the proposed model.

In order to simulate the beginning of failure in a particular network LC, each of them is given a failure probability, p, by using the bathtub curve model [14]. This model splits the lifetime of a product into three stages. First, an early infant mortality failure range is applied to simulate the introduction of the product into the market. Then, a second stage of random failures with constant rate that refers to its useful life. Finally, the probability of failure increases again as the product exceeds its designed lifetime.

Network links are first divided into two groups: High Failure Links and Low Failure Links. For each link in the network [15], the bathtub curve is first applied to determine if it is working properly (state 1) or it started failing (state 2). To do so, a random number r is assessed based on an uniform distribution. If $r < p$, then the link card is labeled as *failing* and the *packet loss function*, F, is applied.

The packet loss function models the impact of a failing line card l on the network at TS t, and is defined as follows:

$$F(l,t) = \min\{\alpha U(l,t) + (1-\alpha) f_i(t - t_f(l), \delta_t), 1\} \quad (2)$$

In eq. 2, term $\delta_t > 0$ represents the maximum lifespan for a line card, i.e., the number of TSs it will be working after it started failing and before finally dying. The hardware deterioration function is given by f_i and represents the packet loss probability due to hardware degradation as a function of the time for a LC that has started failing. Assuming that the LC starts failing at time $t = 0$, we have that a hardware deterioration function must respect the following constraints: i) $f_i(0, \delta_t) = 0$, ii) $f_i(\delta_t, \delta_t) = 1$, and iii) $\forall t \in [0, \delta_t], f_i(t, \delta_t) \in [0, 1]$.

The term $U(l,t)$ is a function that describes the probability to lose packets due to link congestion. Finally, $\alpha \in [0,1]$ is a parameter that balances the two terms of the equation, moving the emphasis of the function from the packet loss due to link utilization to the packet loss caused by the hardware deterioration. It is worth saying that any mathematical function will work as a hardware deterioration function. Therefore, NFS is able to simulate different types of line card deterioration models due to the broad existing market.

B. Simulation set-up

In the following, the input data used for the NFS application is described. The topology that has been considered is Nobel, which is composed of 17 nodes and 52 links. We consider a single line card per link. Regarding the traffic pattern, a set of one-day span traffic matrices belonging to a month have been taken as input from [12]. Concerning the parameters required by the packet loss function, i.e., δ_t, f_i, α (eq. 2), different values have been determined for each of them, with the aim of producing a big data set of combinations that potentially represent a variety of behaviours for LCs, as well as for deterioration rates. In particular, $\delta_t \in \{1, 2, 4, 7\}$ [days], $\alpha \in [0,1] \in \mathbb{R}$, and 4 f_i function candidates (arcsin, logarithm, quadratic, root).

Regarding the predictors, different non-linear functions have been used, which can be split into three groups: classic polynomial predictors, exponential predictors and logarithmic predictors. These predictors allow the system to fit data as close as possible and, therefore, improve the overall performance. The more varied predictors are used, the more probable it is to find a good fit for the data. Data has been split into different prediction environments, each of which has the same values for α and f_i. Moreover, a subset is also added to be able to predict regardless of f_i function, i.e., by being trained only on the basis of α. This last case is referred to as *generic*.

C. Experimental Results

In order to evaluate our proposed LFP framework, two performance analyses have been carried out. Firstly, the overall accuracy of LFP has been tested. Secondly, an additional binary accuracy test of LFP integrated with a rerouting system has also been carried out.

A modified version of NFS comparing the remaining lifetime of the network LC, calculated using eq. 3, with the output of the prediction system, has been used. This modified version lets us get a good estimate of the error by simulating 25 times each of the proposed models described in Sec. II-A.

$$LT(l, t, \delta_t) = \delta_t - (t - t_f(l)) \quad (3)$$

The first proposed analysis studies the Root Mean Square (RMS) error, calculated using eq. 4, as a function of the α parameter for different predictors. The main outcome of this analysis is that the proposed LFP algorithm is unaware with respect to the value of α. In fact, as it can be seen from the results reported in Fig. 2, all the lines do not deviate too much from the average value for different values of α. This is a crucial property for the proposed framework, since the actual value of α is unknown and might depend from several factors (type of damage, entity of the damage, etc.) and changes over time. Moreover, the overall precision of LFP is between 2.3748 TSs and 1.7816 TSs. The standard deviation ranges between 1.4830 and 0.5830.

$$RMS = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2} = \sqrt{2J(\theta)} \quad (4)$$

A second analysis has been performed to evaluate the impact of LFP on a dynamic re-routing system. For this analysis, a threshold θ is set to compare the predicted lifetime of

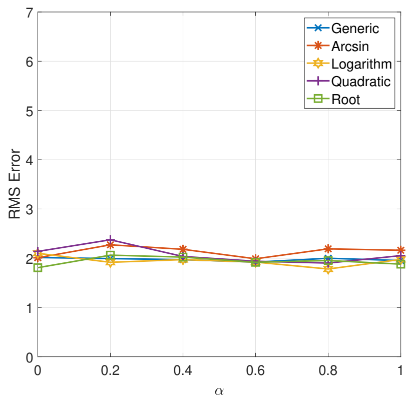

Figure 2. RMS error values for all prediction systems.

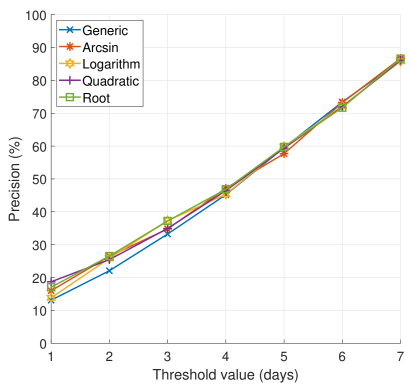

Figure 3. Precision results of the re-routing system.

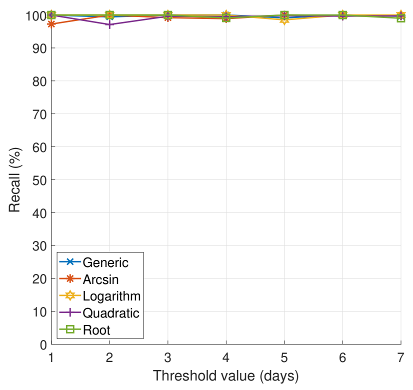

Figure 4. Recall results of the re-routing system.

LCs with. For each failing LC, its lifetime LT is predicted according to eq. 3 on every TS and compared with θ. If $LT > \theta$, no action is taken. On the contrary, if $LT < \theta$, network traffic is re-routed. Threshold values have been set to $\theta \in [1,7] \in \mathbb{N}$. Figs. 3 and 4 report the *precision* and the *recall* of the obtained results, respectively. The precision provides a measure of how many, among all the LCs predicted as failed, have actually failed. The recall is a measure of how many, among the failed LCs, have been correctly predicted. In case of LFP algorithm, recall is a crucial parameter. In fact, a poor recall means that some failing line cards are not correctly predicted (false negative). It implies a high packet loss due to the non activation of the fast re-route mechanism to by-pass the failed LCs. On the contrary, a poor precision, meaning that some LCs predicted as failing are actually working properly (false positives), can be tolerable. In this case, some traffic flows might be unnecessarily re-routed.

Results reported in Figs. 3 and 4 show that conservative thresholds perform better. Specifically, from Fig. 3 it can be seen that the precision monotonically increases with the value of the threshold. The highest precision value obtained during the tests is approximately 85%, meaning that about in the 15% of cases the fast re-route mechanism is unnecessarily triggered. Interestingly, from Fig. 4, it can be seen that the recall is unaware of the value of the threshold, and very close to 100%. It means that the percentage of false negatives is negligible, making LFP algorithm applicable in real network scenarios.

To summarize, the presented performance evaluation has shown that LFP algorithm can replace classical failure detection methods, allowing to reduce the packet loss thanks to its ability to predict failures. Specifically, it is able to detect almost all the failing LCs (recall close to 100%), at the cost of doing some unnecessary re-routing (precision close to 85%).

Finally, a profiling test has also been performed in order to test the time LFP needs to predict. These tests have been performed on a machine with an Intel i7 7700HQ CPU, 8GB of RAM and a NVIDIA GeForce GTX 1050 GPU. To do so, the prediction function has been run 5000 times using the most complex prediction system on randomly generated data.

Table I
PROFILING RESULTS.

Function	# of calls	Execution Time [s.]	Execution time per call [ms.]
Full process	5000	346.058	69.2116
Generation of non-linear features	5000	1.053	0.2106
Prediction	5000	345.006	69.0012

As seen on Tab. I, the full process took 346.058 s., which means that a single prediction with the most complex model takes an average of 69.2116 ms. to complete, a very short time that benefits its integration with other systems.

IV. Conclusion and Future Works

This paper proposes an ML-based framework to predict the remaining lifetime of network LCs. Such prediction can be used as a triggering event for the activation of a fast re-routing mechanism which would lead to reduce the impact on QoS performance. A LC failure model has been defined as a part of the proposed framework, which has been evaluated on a real network with varying traffic. Results show that our ML-based framework can be applied to the problem of predicting the lifetime of network LCs with approximately 100% of recall and a good precision. As future work, we have considered to add new features to the system and use neural networks adapted for regression [7]. Moreover, we are currently working on integrating the proposed solution into a POX-based SDN controller.

Acknowledgements

This work has been partially funded by the 4IE+ Project (0499-4IE-PLUS-4-E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Spanish Ministry of Science, Innovation and Universities (RTI2018-094591-B-I00), by the Department of Economy and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] P. E. Heegaard and K. S. Trivedi, “Network survivability modeling,” *Computer Networks*, vol. 53, no. 8, pp. 1215–1234, 2009.

[2] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 2013, p. 15.

[3] S. S. Ahuja, S. Ramasubramanian, and M. M. Krunz, "Single-link failure detection in all-optical networks using monitoring cycles and paths," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1080–1093, 2009.

[4] F. Aubry, D. Lebrun, S. Vissicchio, M. T. Khong, Y. Deville, and O. Bonaventure, "Scmon: Leveraging segment routing to improve network monitoring," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[5] P. L. Ventre, S. Salsano, M. Polverini, A. C. A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: A comprehensive survey of research activities, standardization efforts and implementation results," *arXiv preprint arXiv:1904.03471*, 2019.

[6] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Computing Surveys (CSUR)*, vol. 42, no. 3, p. 10, 2010.

[7] D. F. Specht, "A general regression neural network," *IEEE Transactions on Neural Networks*, vol. 2, no. 6, pp. 568–576, Nov 1991.

[8] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[9] A. Ng, "Machine Learning," Course by Stanford University on Coursera.

[10] B. G. Asseffa and O. Ozkasap, "Machine learning framework for traffic aware energy efficient routing in sdn," *IEEE 16th International Conference on Dependable, Autonomic & Secure Computing*, 2018.

[11] R. Fletcher, *Practical Methods of Optimization*. Wiley, 2013.

[12] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.

[13] M. Peuster, S. Schneider, and H. Karl, "The softwarised network data zoo," *arXiv preprint arXiv:1905.04962*, 2019.

[14] G. Klukte, P. C. Kiessler, and M. A. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on Reliability 52*, 2003.

[15] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Ganjali, and C. Diot, "Characterization of Failures in an Operational IP Backbone Network," *IEEE/ACM Transactions on Networking*, 2008.