 TSG IT Systems

Proof of Concept:

Machine-Learning-based Link Fault Detection

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Version | Author | Validation status | Checker |
| 04.08.2020 | 1.0 | Samuel Guerchonovitch | Not validated | L. Schvartser |

Summary

[1. Introduction 3](#_Toc37270035)

[1.1. Purpose of the study 3](#_Toc37270036)

[1.2. Context 3](#_Toc37270037)

[1.3. Assumptions 3](#_Toc37270038)

[1.4. Definitions 4](#_Toc37270039)

[2. Principle and Models 4](#_Toc37270040)

[2.1 Why using machine-learning? 4](#_Toc37270041)

[2.1.1 Provides a more efficient way of detection 4](#_Toc37270042)

[2.1.2 Identify when a new node is deployed in the network 5](#_Toc37270043)

[2.1.3 Predict the impact of the integration of a node in the network 5](#_Toc37270044)

[2.2 Which data to be used? 5](#_Toc37270045)

[2.3 Supervised Learning models and Measurements of choice 6](#_Toc37270046)

[3. First proof with a very simple model: the 5-nodes model 7](#_Toc37270047)

[3.1 Model presentation 7](#_Toc37270050)

[3.2 Experiment 8](#_Toc37270051)

[3.3 Results 9](#_Toc37270052)

[4. To go further with bigger network and scripting 12](#_Toc37270053)

[4.1 Extract the data out of GN Software 12](#_Toc37270055)

[4.2 Capturing data 14](#_Toc37270060)

[4.3 Integrate the python script and the scp service in the GN Software 14](#_Toc37270061)

[4.4 Results? 14](#_Toc37270062)

[5. Conclusion 16](#_Toc37270063)

1. Introduction

# Purpose of the study

The purpose of this study is to find a way to predict with a low cost a link fault in a complex network, changing its shift very quickly. The main idea is to be able, by watching the flows in the network and knowing the initial state of the topology, to determine the entrance or the fault of a link in this network by simply monitoring.

For this aim, we would like to verify in which situation the model is realizable:

* How is the model doing in a very simple experiment?
* How would we exploit this technology in the client environment?
* What do we need to deploy for the model to work?
* How scalable is the model?

# Context

The client is in demand for a more efficient way of dealing with their IoT architecture, which by definition is changing very fast on time. For this purpose, it is crucial to:

* Understand when and where is a fault appearing, in order to deploy an immediate correction,
* Understand how the model is facing the change in the topology due to the integration of a new node,
* Be able to predict the impact of a new node integrating the current topology.

It is in this context that this report has directed its purpose.

As we are now in a very first PoC, we are not aiming at testing the model on the real network, as it would mean activity discrepancy for the client. Therefore, we must use a lab version first, which implies several assumptions.

# Assumptions

For the study, due to the simplification implied by the lab, we did the following assumptions which would need a specific focus in case the technology is rose to further tests:

* All the nodes used in lab are virtual machines, built the same way, depending on a personal computer. It supported very well more than 20 nodes, but we should be aware that further testing must be done on powerful server able to produce 60 nodes or more.
* All the nodes in the lab are virtual over a logical machine. Therefore, their behavior is necessarily different from a behavior of several physical routers of different age, craftmanship and use. In lab, the performance of the nodes depends strongly on the performance of the computer.
* In the first try, the monitoring has been done hands-on. In the last part, we tried to industrialize it as it is important to have more scalable methodology as soon as we increase the number of nodes in real life.
* We are assuming that only one link fault is occurring at a time. As the model may, on term, deal with the scale of the microsecond, modeling a double link fault at the microsecond seems rare enough to consider this assumption valid.

# Definitions

|  |  |
| --- | --- |
| Definition | Description |
| Supervised Learning models | Data science models which tries to label observations based on similarities with already labeled data.  It is opposed to unsupervised learning, such as clustering models, which are grouping observations into unnamed classes.  These specific models are the most understandable as the labels and easiest to stick to reality in non-changing models. Its main drawback is that **it cannot consider completely new classes** if it has not been observed beforehand**.** |
| TTL  (Time to Live) | Also known as hop limit. This mechanism is usually defining the lifetime of data in a network. It counter regular issues of loops, but also to count the number of hops encountered before reaching the destination. |
| SDN  (Software Defined Network) | Technology based on **OpenFlow**. The software provides an efficient way of virtualizing network nodes (such as routers or switches) in order to **test coherence and performance events of a network**. It is also a good tool to practice a network architecture before deployment. |
| OSPF  (Open Shortest Path First) | Routing protocol for IP networks. The routers are addressing to their neighbor to map the area. An area is defined as **Backbone (area 0)** and all the future areas are connected (physically or virtually) to the Backbone. Each area defines its **shortest path** thanks to graph algorithms such as Dijkstra to forward packets. |
| Pandas package | Pandas is a strong Data Science Python package, providing all the necessary functions for data engineering. It arranges the data into data frames, which is a Numpy structure including non-numerical values.  Pandas is must-have for almost all the main data science models in python, which are customized to work with Pandas Data Frames. |

1. Principle and Models

# Why using machine-learning?

## Provides a more efficient way of detection

A network architecture today is already able to make the nodes communicate with each other, thus understanding when a neighbor is faulty. Protocols already exist in order to identify and report it. However, in a very changing network such as in an IoT network, such information faces two main drawbacks:

* The time of identification is very long in the scale of this kind of network (expect 1 to 3 seconds to react), which is not affordable when a correction is necessary,
* The extra packets sent in the network implies a lot of traffic, which leads to latency. In such an environment sensitive to latency, it is not acceptable.

The Machine-Learning based model is proposing a more “passive” way of getting the information. Instead of actively asking each node about their neighbors’ status, we can monitor the nodes directly, centralizing the information into one *monitor* node, with the least information possible. It can then request immediate reaction in case of fault.

According to **[1],** the gain in latency would be of the scale of 10 000 times faster, by avoiding those two issues at the same time.

It would then answer the first requirement of the client.

## Identify when a new node is deployed in the network

The model is not defined to identify a new node coming into the topology. As the model is supervised and classification-based, the label of the fault must be explicitly defined. It is then impossible to identify a label “Node added after node X” without experimenting and observing it beforehand.

It seems impossible at first sight to test all the possibilities at any time in the network in order to predict a possible node addition.

The only thing it can provide is:

* Detecting and Localizing a fault in the network
* Detect if the node has been replaced and if the network is back in its previous state, or if a new fault is detected

Therefore, the model as is cannot provide a result for the second requirement of the client.

However, it may be a base of reflection on how to discover the network. For instance, we can think of a model where we label specific structures in the network and prepare the scenario, hence making the model evolve into something more.

If we isolate the communication threads coming only from these nodes, we might be able to identify one of the scenarios we have tested on it.

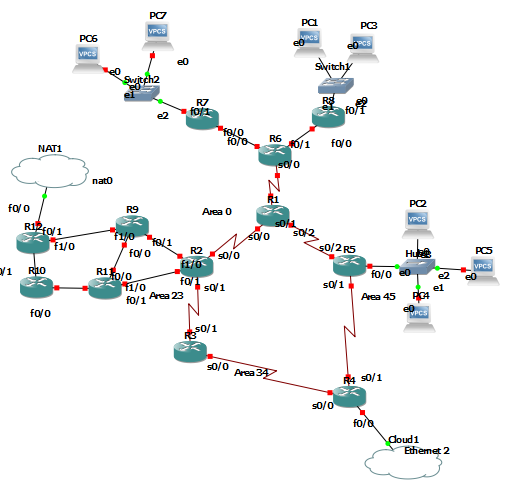
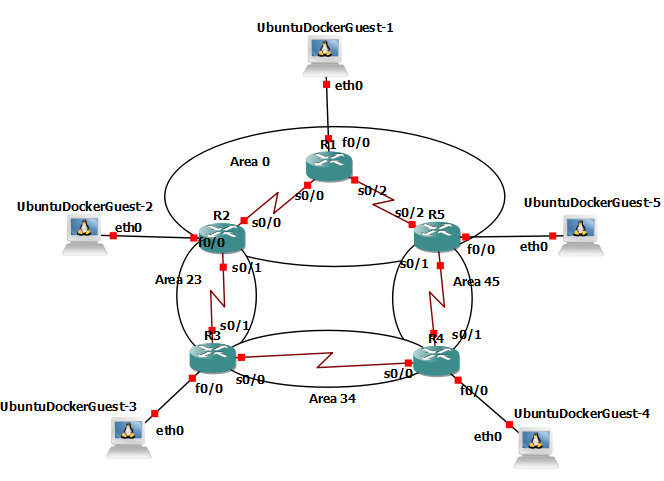
 

Fig. 1 - Detecting a loop we recognize with labeled node integration scenario

## Predict the impact of the integration of a node in the network

The question beneath it is: is it possible to reverse the model? Based on our results, how can we predict the observations?

Unfortunately, the model has no reason to be bijective: for a specific result, we can expect several possible observations.

Finding a way to predict this would ask for a new model to think about.

Therefore, **the model cannot provide a result for the third requirement of the client.**

# Which data to be used?

As the paper suggested, we can use the most basic information a capture of the network would provide:

* **Packet Flow Rate** is the most important information, as we are expecting the flow rate to change according to the dynamic routing.
* **Delay** is also an important feature as a different route may lead to different cost in delay.
* **Packet Loss** can be a good way of identifying link drop event as the packets sent during the route recalculation will eventually be dropped and resent in most cases. It can then help the model discriminate between the real faults and other changes in the traffic (such as a usage peak).

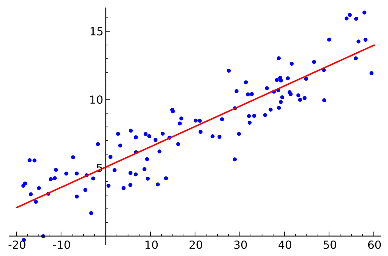
If the study led to a try on more nodes, and if accuracy were to be affected, we also suggested the following features:

* **Time to Live** may have a pretty high entropy when dealing with dynamic routing. If TTL changes in average of more than 1, we can expect that at least 1 hop is added in the routing. Hence avoiding the case where the delay remains slightly the same, but the route is changing.

# Supervised Learning models and Measurements of choice

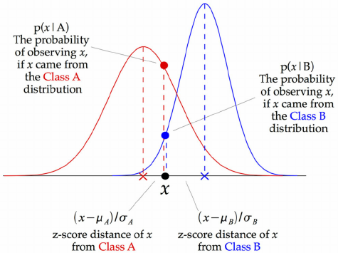
In this PoC, we will try 4 Supervised classification models:

* **Logistic Regression**, which is the basic way of classifying data. The main idea beneath this model is a simple regression to decide whether the observation is more probable to be of class A or class B

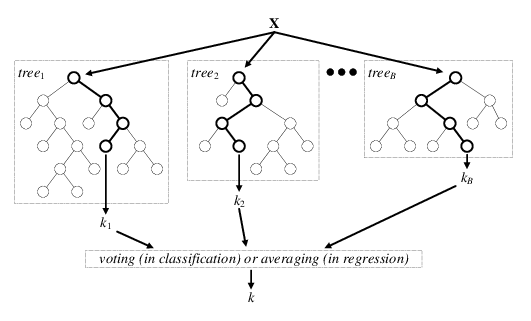


It is the most basic classification method and will be our reference to value the other models.

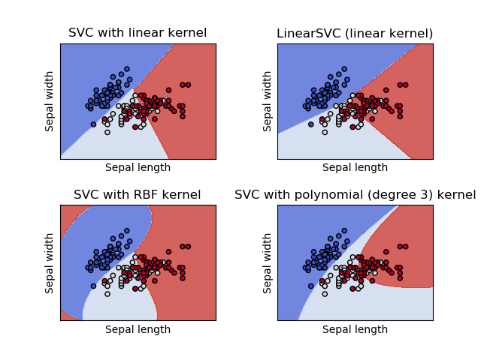
* **Naïve Bayes, more particularly Gaussian NB** model, which can be interesting when we try to define the probability for an event A to occur knowing an event B occurred and the probability of event A occurring knowing event B. For that, we have to assume the event B is of some probabilistic distribution. As we modeled or boost with Gaussian distribution, we will go with Gaussian NB.



* ***Random Forest***, is an ensemble model (based on decision trees), but instead of using only one decision tree, very sensitive on the bias of the input, reproduce the experiment with a large set of decision trees, using a sample of the observations. By cross-validation, all the decision trees vote for one final class. This method is very promising because fast and independent of the scale of the data.



* **SVM** is also a promising model: instead of a line to confront each feature as Log.Reg. does, it defines the hyperplane (not necessarily linear) separating the classes. It can then do more fitted “cuts” in the data distribution in its dimensions.



We are focusing on two main measures: the F1-Score and Accuracy.

The **F1-Score** considers equally both the Precision and Recall:

We are not focusing on either the Recall or the Precision because their importance is equal:

* The Precision is measuring how minimal are the false positives (the faults detected, but not occurring).
* The Recall is measuring how minimal are the false negatives (the faults occurring, but not detected)

Both cases are as important as the other.

The **Accuracy** gives an overall measurement of the good predictions:

It is a very basic way of measuring, efficient only in balanced data (which is the case in our study, as we decide how much data we collect).

1. First proof with a very simple model: the 5-nodes model

# Model presentation

As our first model was a try hands-on, we wanted our model to be the simplest possible, but also presenting the first challenges for the machine learning model.

We chose the 5-nodes loop model as seen below:

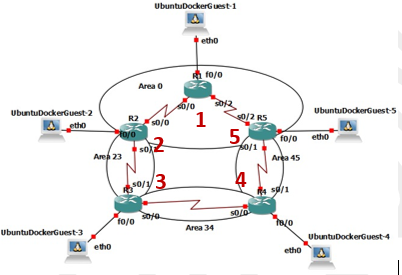


Fig. 2 - 5-nodes loop model

With this model, we can add two new assumptions:

* In lab, we can suppose that the role of each node is symmetrical: *node 1 will behave for node 2 as node 3 will for node 4.*
* We can suppose that each node being the same, same behavior means same properties (flow rate, delay, loss…) in a symmetrical scenario: *cutting the 1-5 link will trigger the same result for node 3 than cutting link 3-4 for node 1*

With these two assumptions, we can reduce our scope of study to one node and assume the same happens to the others (which is verifiable with experiment).

Even if the model is simpler, we still have values changing significantly with a fault as we can presume with the following scheme:

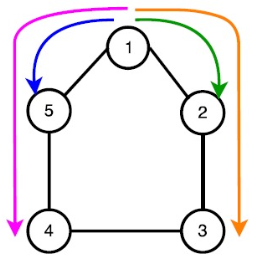
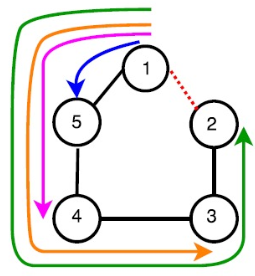
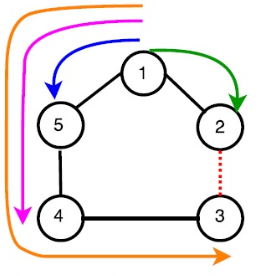
  

Fig. 3 - Normal routing from node 1 Fig. 4 - Routing with link 1-2 fault Fig. 5 - Routing with link 2-3 fault

# Experiment

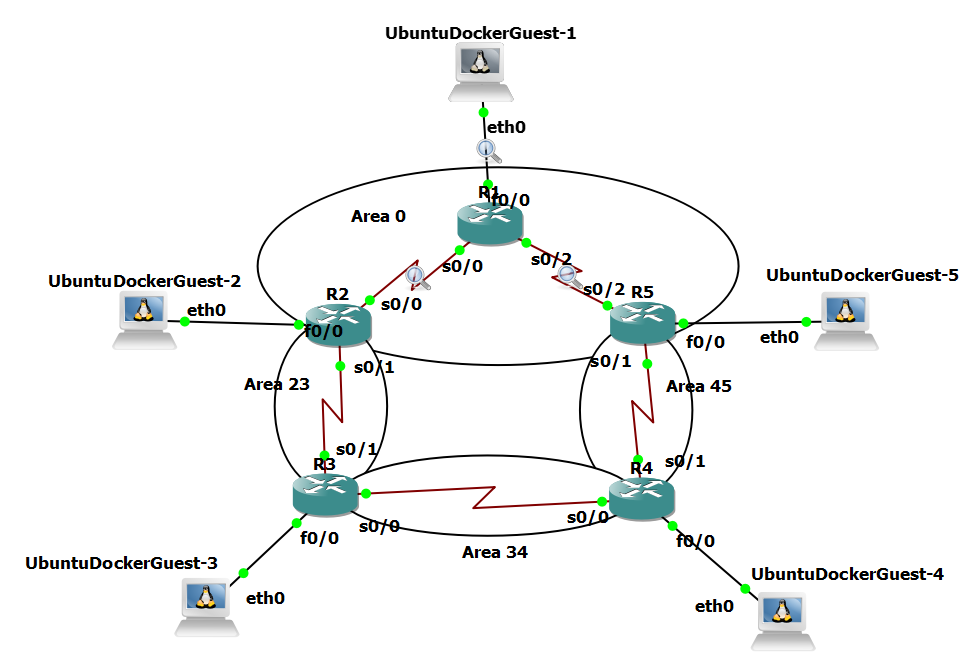


Fig. 6 - 5-nodes loop model in GNS3

As we have no access to physical machines, we used GNS3 as an SDN tool to simulate network nodes, mainly Cisco routers and Docker containers containing Ubuntu for parallel requests. We capture traffic around the node 1 and do our stats.

We deployed the OSPF protocol for dynamic routing between the routers, dealing with the redirection of the flow in the link fault scenario. A pro version of VMWare is used for the virtual machines in Linux.

A screenshot of a cell phone

Description automatically generatedWe then request a ping from every endpoint to every other endpoint:

When the network becomes stable, we identify the delay encountered by the ping in the red circles. We can then establish a mean delay per request per destination.

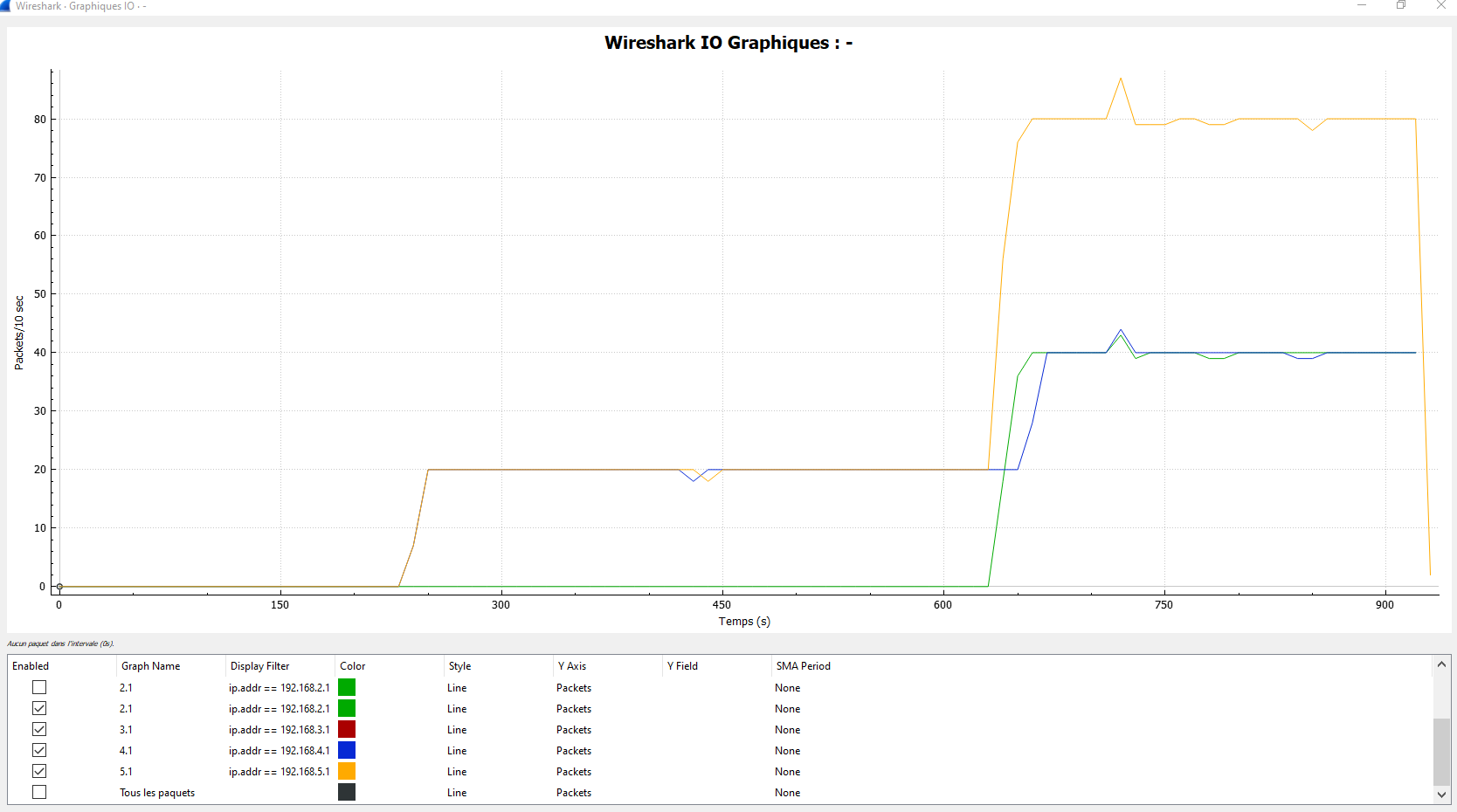


Fig. 7 - Graphique I/O of node 1

Wireshark provides some way of observing the flow rate, which allows to pick the values for each destination.

The last values to get is the number of packets lost at the endpoint 1:

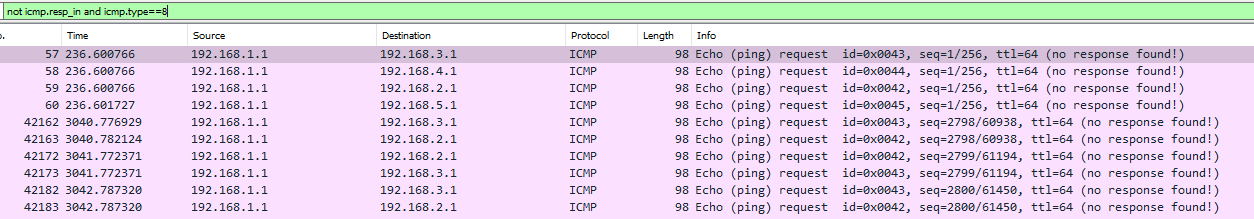
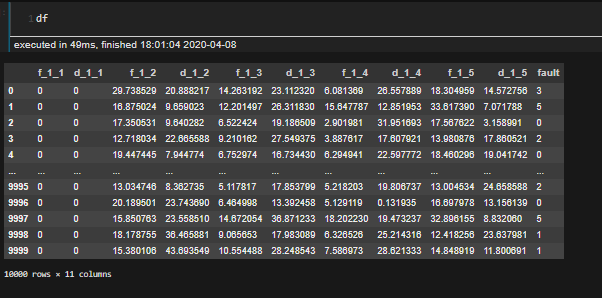


Fig. 8 - Filter to get the lost packets (having no response)

With this filter, we can identify the loss by hand and determine the last values. We realized this last value is not necessary for such a model as the rest is discriminative enough. However, it has to be used for more complex networks.

# Results

With this hands-on method, by observing the shape of the data, we were able to collect data and boost our observations with normal distributed data:



We can recognize in each line an observation, with the 10 features (2 per destination) and the label *fault*.

The results were the following:

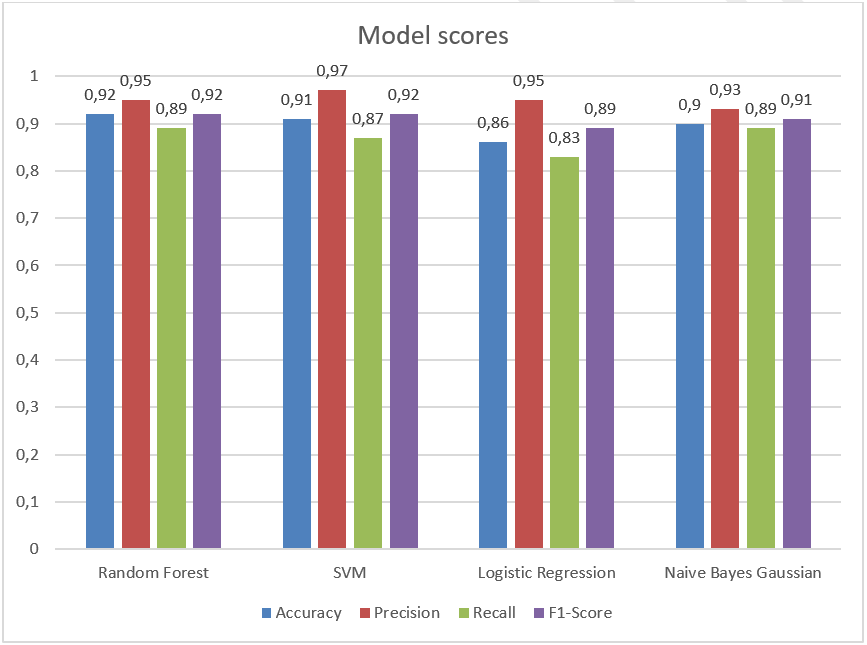
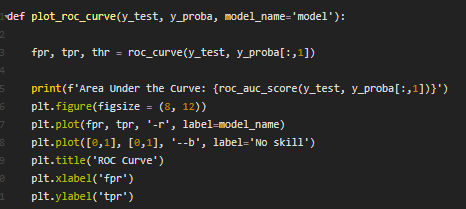


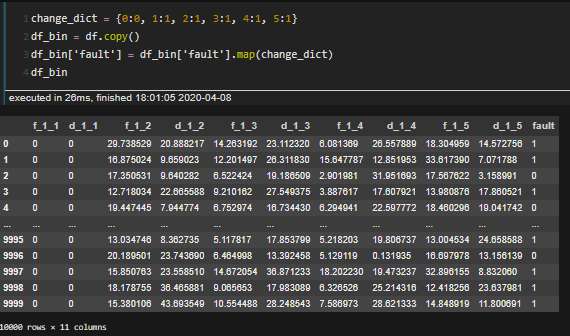
Fig. 9 - Scores per ML models

As we are focusing mainly on the F1-Score and secondly on Accuracy, we can verify the best results are those of the Random Forest model, not far is the SVM model.

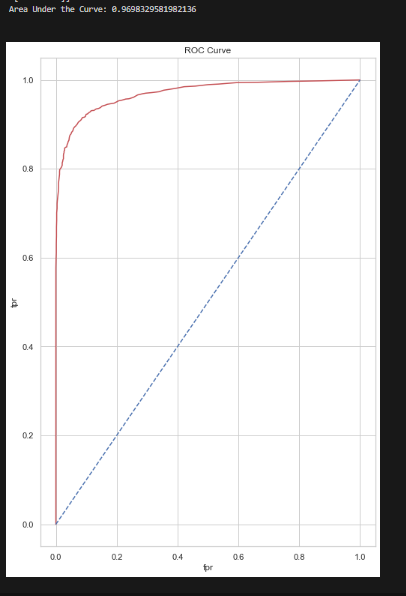
To visualize how the trade off between Recall and Precision occurs, we can look at the ROC curve, and more precisely at the area under this curve:



The ROC curve can only be used on binary class (0 or 1, faulty or not faulty), hence needing to binarize our data frame:



We have the following result with the Random Forest model:



We can see that the curve is strongly pinning up to the top left side, which means the trade off between Precision and Recall toward a threshold is not a constraint for this model. The AUC is almost 97% of the [0,1] x [0,1] square.

As expected, we verified this very simple model is enough to identify and localize the link faults with a very high accuracy. Only two of the recommended features were enough to reach a good result.

1. To go further with bigger network and scripting

As a reminder, we are aiming to scale the experience on more complex, not symmetrical and node-independent network structure. Therefore, we can’t afford to continue working hands-on. The following has been deployed:

* The data capture must be done on the node itself instead of observing its *In/Out* transfers,
* The monitoring of the data must be centralized and sent regularly
* The network must be more realistic, simulating *“real”* conversations (TCP conversations for example) instead of ICMP messages such as the pings

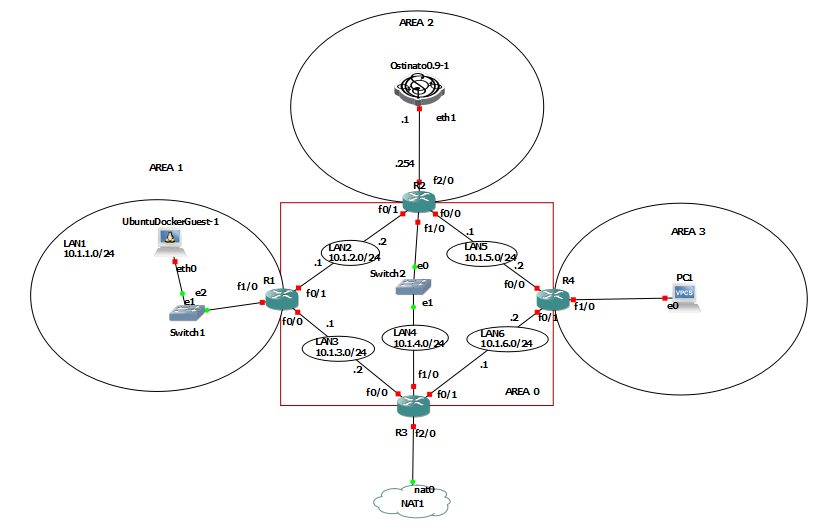


Fig. 10 - A more complex, non-symmetric architecture

In order to tackle this issue, we need to face three challenges in our GNS3 tool:



# Extract the data out of GN Software



GNS3 is not providing any link between the virtual machines built in the virtual network and the real machine supporting the VMs. This is due to the fact that GNS3 is the intermediate layer between the OS and the emulated machines:

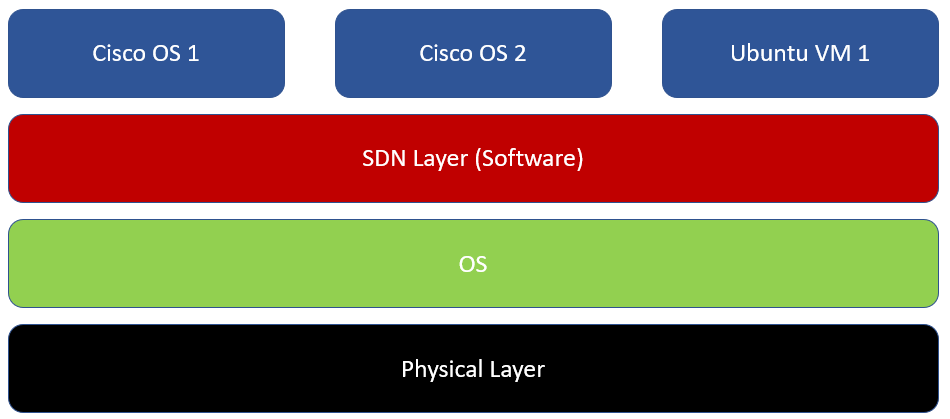


Fig. 11 - Representation of GNS3 (SDN-based) emulation

A workaround has been to extract the data on the fly after collecting it. For this, we recommend the following via Python:

* Transform the data into pickle or csv (using the *Pandas* package for instance)
* Transfer the data through the internet to your local machine via SCP, as GNS3 allows internet connection via **NAT**. For this aim, we use the python packages called *scp* and *paramiko*.

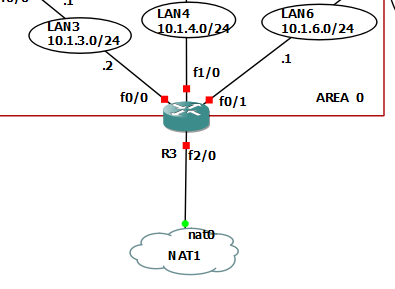


Fig. 12 - GNS3 allows connection to internet via NAT

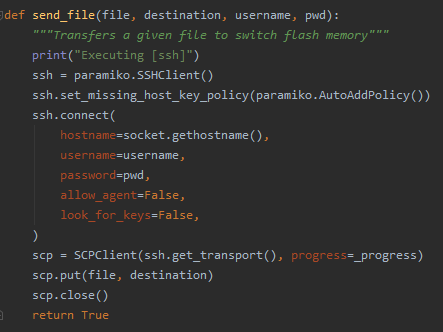


Fig. 13 - An example of use of scp and paramiko package

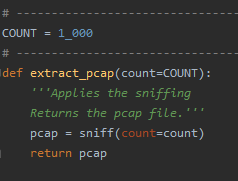


# Capturing data

To monitor data, we need to sniff the network on a node and work on the resulting pcap file.

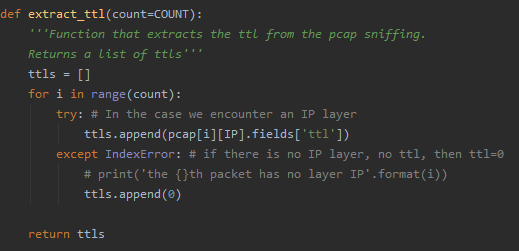
To do so, we worked with the python package called scapy. It allows very agile actions such as:

* Sniffing up to a specified number of packets and return a pcap file:



Since it is captured, every packet is a pcap object.

* It then allows to access the different properties of these packets which can be of interest. In the following example, we access the ttl of all the sniffed packets:



With this powerful tool, we can fast create list of objects to restructure into data frames, csv files or pickle files.

# Integrate the python script and the scp service in the GN Software

GNS3 allows the use of Docker containers, we are then building the docker container on a Virtual Machine and running directly the script on start.

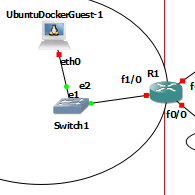


Fig. 14 - Ubuntu container with script embedded in the network

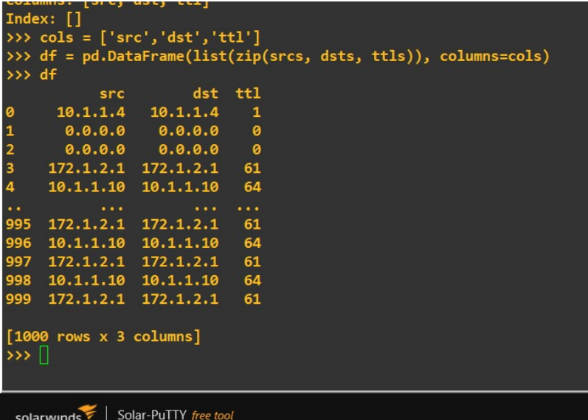


Fig. 15 - Example of script running on the ubuntu container

# Results?

As outside events occurred during this study, we had to work in degraded mode. Therefore, we did not finish to parametrize correctly the scp transfer.

Apart from this issue, everything is set and running.

We then will need to:

* Test it on the presented architecture
* Stress test it on more complex architectures (with 30/60 nodes)
* Test it on an architecture closer to the client’s.

In addition, non-Free tools can be very efficient in our case. We should consider paying a monitoring tool license for instance with NPM or another competitor.

1. Conclusion

The Machine Learning based technology to detect and localize the link faults in IoT architectures is promising to answer to first requirements of the client.

However, the POC needs to go further with the experiments listed above before presenting to the final client.

This study would have provided with:

* A first positive Proof of Concept,
* Data Science models and measures to follow,
* Method of operation to follow
* A script to run in a container

1. References

[1] *TE-Based Machine Learning Techniques for Link Fault Localization in Complex Networks*,

Srinikethan Madapuzi, Tram truong-Huu, Mohan Gurusamy, NU Singapore (2016)

[2] *Machine Learning-based Link Fault Identification and Localization in Complex Networks*,

Srinikethan Madapuzi, Tram truong-Huu, Mohan Gurusamy, NU Singapore (2019)

[3] Scapy project,

<https://scapy.net/>

[4] Sklearn project,

<https://scikit-learn.org/stable/>