Predicting Traffic Patterns in Software Defined Networks

Liva Giovanni - liva.giovanni@spes.uniud.it
Hermann Hellwagner - hermann.hellwagner@itec.uni-klu.ac.at
Marino Miculan - marino.miculan@uniud.it

May 28, 2015

Abstract

Just the prediction part

1 Introduction

The predictability of network traffic is the main aim of the thesis. Usually, there are two different category of network prediction: short and long period predictions. The short forecast is used to guess values in terms of seconds or minutes. Instead, the long one is adopted to estimate the future workload. Therefore, it favors the possibility to produce better planning and decision. To be able to predict future load of a network, we have to create a model of its behaviors. On the changing of a model we have different characteristics such as the correctness of the prediction and its adaptability.

There are two type of models: **Supervised** and **Unsupervised**. The *Supervised* algorithms takes as a input a set of objects and the desired output. The set of objects it is called training data. The learning algorithm analyzes the training data and produces an inferred function that its behavior is checked with the last input. The internal structure of the model is changed according to the error between the forecast and the desired result. Instead, the *Unsupervised* learning tries to find hidden structure in unlabeled data. The difference with the *Supervised* learning algorithms is that there is no error or reward signal to evaluate a potential solution.

We focus over the long term prediction and only on supervised classifier. The decision of which classifier chose has been taken conducting an experiment in a small simulated network. We simulate a normal scenario of daily network usage through a network of 4 nodes. We repeat the simulation thirty times collecting at each execution statistics of the links utilization in terms of network bandwidth and the load of the switches. From this information, we have created different dataset changing the features used and the numbers of the last observations. Then, we have tested the prediction precision and recall of different algorithm at the varying of the distinct datasets.

The implementation of the prediction is designed in four different phases

- Observer
- Analyzing
- Plan
- Execute

A graphical representation how the four module are interconnected is given in Figure 1.

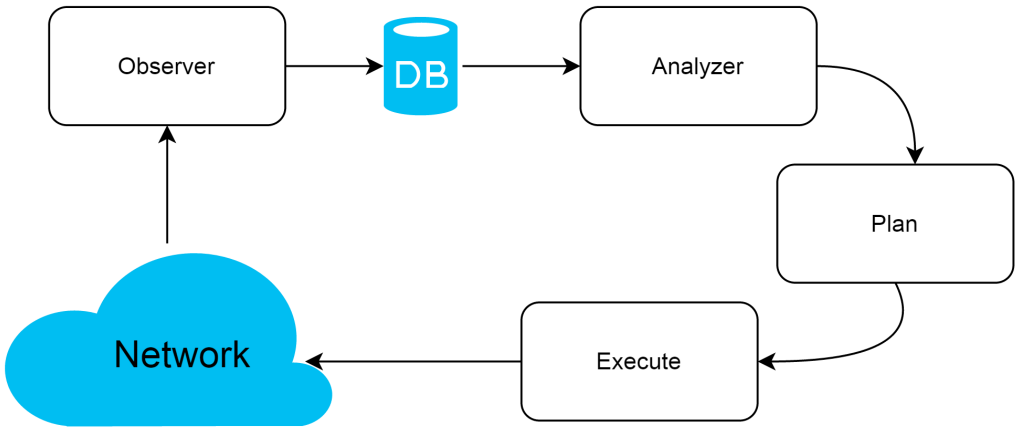

Figure 1: The different phases of the prediction

The first module, **Observer**, is implemented as a daemon in the cloud application. Every few minute it launches a python script which queries the network controller and then stores the result inside the database. The information saved regards the network load, switches and flows.

The second one, **Analyzing**, is done looking through the information inside the database. A java application collects the data from the database and converts the knowledge in the Attribute-Relation File Format (ARFF). This format is an ASCII text file that describes a list of instances sharing a set of attributes. The application depends on the *weka* (Waikato Environment for Knowledge Analysis) package, a well known suite of learning machine algorithms developed by the University of Waikato. The ARFF files are read by the weka package and used to produce the model that is adopted to make predictions.

The third element, **Plan**, is demanded to the Administrator. He or she can write rules to specify what to do when a particular event occurs. In the cloud application the administrator has the possibility to create the rules that are stored inside the FloodLight controller.

The last phase, **Execute**, is implemented by the controller. It monitors the network and every few minutes it makes predictions using the previously generated model. When it perceives from a forecast that some rules can be applied, it fires them.

Every module is uncoupled from the others. This design decision of modularity gives us the possibility to change or upgrade every module whenever

there is the necessity. This feature is crucial for the prediction phase. We can test new classifiers or the addiction of new features in a separate and controlled network without affecting the production one. Moreover, we can hot swapping the model using the cloud interface. We have designed the controller to work with a different model for each switch. This decision brings the possibility to predict when a particular node will be overloaded with more precision and recall.

2 Prediction over the Network

2.1 Configuration

We have set up a small simulation to decide which classifier algorithm works better for our purpose. The candidates were the following:

- *NaiveBayes* [6]: is a classifier based on the Bayes theorem. It assumes independence between the features of the model. It is in the family of simple probabilistic classifiers.
- *SMO* [10, 7, 5]: It is an algorithm that employs the support vector machines. The classifier learns by solving an optimization problem.
- *BayesNet* [2]: It uses the K2 learning algorithm that is in the family of the Naive Bayes. It uses a hill climbing algorithm restricted by an order on the variables of the model.
- *MultilayerPerceptron* [12, 1]: It is the most known application of the neural network that uses the back-propagation technique to adjust its internal structure.
- *J*48: It is a java implementation of the C4.5 [11] decision tree algorithm.
- K^* (*KStar*) [3]: It is an instance-based classifier. It classifies a new instance using the one that is more near to one it has learned in terms of entropy.
- *ZeroR*: This is the dumbest classifier of the list. We have chosen to test it so we can measure other classifiers by how well they do compared

to this minimal level of performance. Given a certain data set, ZeroR permits to find out what is the minimum performance we may expect.

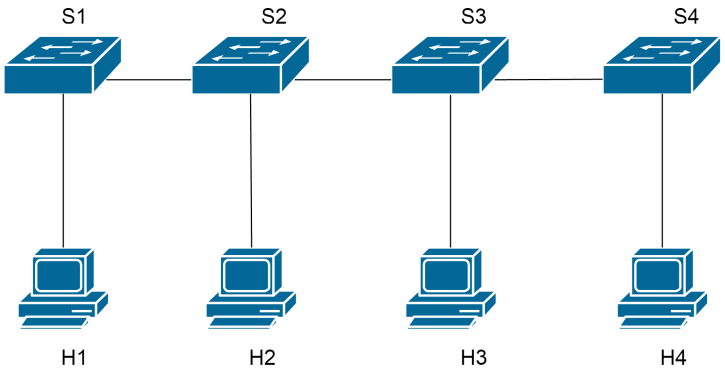

Figure 2: Topology of the network used to test the different classifiers

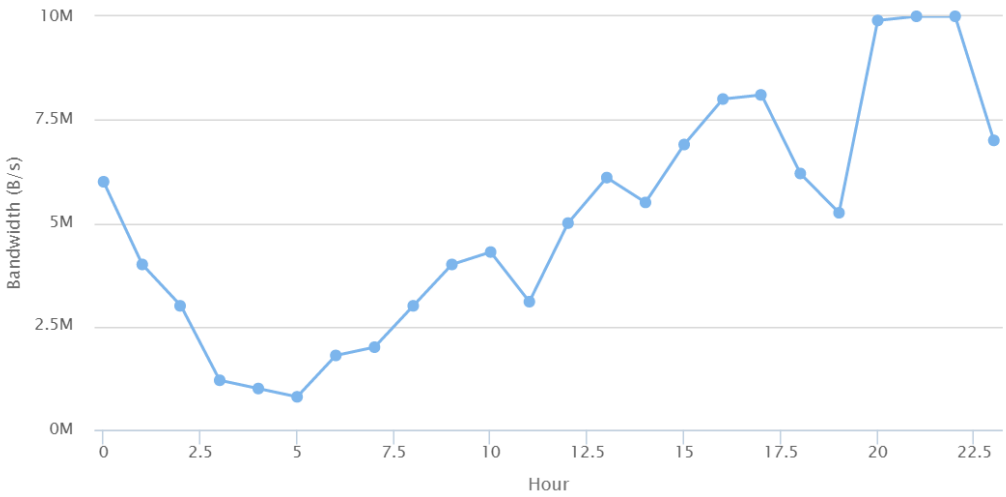

Figure 3: Graph of the bandwidth simulation of a daily usage of the network

To found the most promising classifier, we have created a network of 4 nodes shows in Figure 2. In this network we have generate traffic from the node $H1$ directed to $H4$ for a day respecting to the scenario predefined. The graph bandwidth generated is shown in Figure 3.

The creation of the network relies on the virtualization of the hardware components. Therefore, we have simulated it using Mininet [8] with OpenFlow [9] switches. The chosen controller is FloodLight [4], an open source controller for Software Defined Network. Regarding the Observer we have created a python script that queries the controller saving the information bla bla bla

.

Net -¿ Mininet
Observer -¿ Daemon Analyzer -¿ Weka -¿ Struttura modulare -¿ Cambiare algo quando si vuole Plan -¿ Operator w/ Web Interface config the FloodLight Module Execute -¿ FloodLight Module exe the rules

2.2 DataSet

2.3 Evaluation

classification learning is appropriate for any problem where deducing a classification is useful and the classification is easy to determine.

2.4 Result

Printare i dati in excell, dare qualche valore

2.5 Discussion

Avere 50% di successo -¿ 10 volte meglio di sparare a caso $1/21 = 4.76\%$

References

[1] Eric B Baum. "On the capabilities of multilayer perceptrons". In: *Journal of Complexity* 4.3 (1988), pp. 193–215. ISSN: 0885-064X. DOI: http://dx.doi.org/10.1016/0885-064X(88)90020-9. URL: http://www.sciencedirect.com/science/article/pii/0885064X88900209.

[2] Remco R Bouckaert. *Bayesian network classifiers in weka*. Department of Computer Science, University of Waikato, 2004.

[3] John G. Cleary and Leonard E. Trigg. "K*: An Instance-based Learner Using an Entropic Distance Measure". In: *12th International Conference on Machine Learning*. 1995, pp. 108–114.

[4] *FloodLight*. http://www.projectfloodlight.org/floodlight/. Accessed: 2015-05-25.

[5] Trevor Hastie and Robert Tibshirani. "Classification by Pairwise Coupling". In: *Advances in Neural Information Processing Systems*. Ed. by Michael I. Jordan, Michael J. Kearns, and Sara A. Solla. Vol. 10. MIT Press, 1998.

[6] George H. John and Pat Langley. "Estimating Continuous Distributions in Bayesian Classifiers". In: *Eleventh Conference on Uncertainty in Artificial Intelligence*. San Mateo: Morgan Kaufmann, 1995, pp. 338–345.

[7] S.S. Keerthi et al. "Improvements to Platt's SMO Algorithm for SVM Classifier Design". In: *Neural Computation* 13.3 (2001), pp. 637–649.

[8] *Mininet Network Virtualization*. http://mininet.org/. Accessed: 2015-05-25.

[9] *OpenFlow*. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf. Accessed: 2015-05-25.

[10] J. Platt. "Fast Training of Support Vector Machines using Sequential Minimal Optimization". In: *Advances in Kernel Methods - Support Vector Learning*. Ed. by B. Schoelkopf, C. Burges, and A. Smola. MIT Press, 1998. URL: http://research.microsoft.com/j%CC%83platt/smo.html.

[11] Ross Quinlan. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[12] Dennis W Ruck et al. "The multilayer perceptron as an approximation to a Bayes optimal discriminant function". In: *Neural Networks, IEEE Transactions on* 1.4 (1990), pp. 296–298.