An Overview of Network Troubleshooting With SDN

***Abstract*—TODO**

***Index Terms*—TODO**

I. Introduction

Since years, the main problem in network maintenance is network debugging. Bugs are really difficult to spot with tools like ping, traceroute and SNMP agents, which are the most used to diagnose issues. [**?**]
Hopefully, this could change thanks to the deployment of Software-Defined Networks. SDN is an architecture developped to fix the mess in the control plane. It acts like a logically-centralized controller which manages switches by installing (or uninstalling) rules in them. The controller also read traffic statistics and respond to events. An handler is attached for each events and respond to its event by applying the routine establised by the network engineer.
Thanks to SDN, it should be possible to automate troubleshooting, this will be seen in (TODO: how to reference to a next section ?).

II. SDN layering, the key used to a better troubleshooting

Finding and solving network bugs are not the aim of SDN, but we can use it to re-think the way we troubleshoot networks.
The SDN architecture is decomposed into layers, those layers can be represented in a two dimensionnal array. As you can see on Figure **??**, we have the two main layers called *State layers* and *Code layers*. The state layers hold a representation of the network's configuration for each parts of the network architecture. The code layers implement logic to maintain the mapping between two state layers. Each states layers should verify the equivalence properties, which means that each of them should correctly mapping every other state layer. The idea is that, for each policy, if the state layers are correctly mapped among each other, then the policy is set and acts like it should.
Thus, thanks to the SDN stack, we are able to first build a tool to check consistency between state layers in order to identify on which part of the network architecture a bug is happening.
When the layer is identified, an other tool take over to localize the issue inside the code layer. We will see in (TODO: indicates next section) which kind of tools could be used to handle that.

III. Network Troubleshooting

Expliquer ici par rapport au layer
Reference vers les outils develope dans d'autres articles +

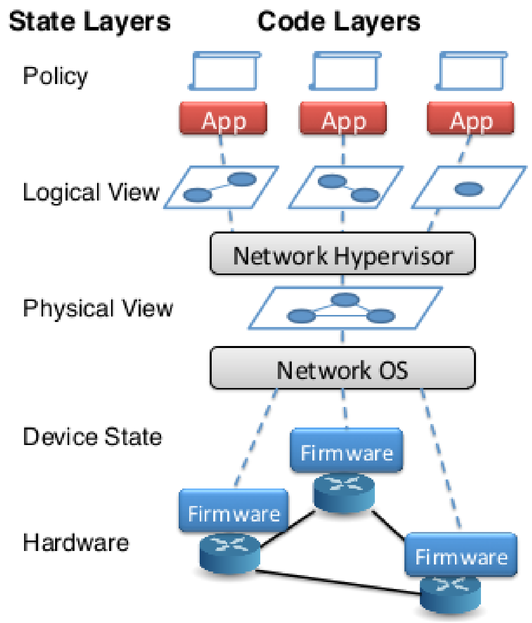

Fig. 1. SDN architecture

explications sommaire de ceux-ci
voir page 2 a Nice way (challenge pour tester l'open flow) et les solutions qui existent)

IV. A

V. Conclusion

TODO

References

[1] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: change you can believe in!" in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070569

[2] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[3] T. Nelson, A. Guha, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "A balance of power: expressive, analyzable controller programming," in *HotSDN*, 2013, pp. 79–84. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491201

[4] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491186

[5] J. Rexford, “Programming languages for programmable networks,” in *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL ’12. New York, NY, USA: ACM, 2012, pp. 215–216. [Online]. Available: http://doi.acm.org/10.1145/2103656.2103683