Towards meticulous data plane monitoring

Apoorv Shukla[†] Said Jawad Saidi[†] Stefan Schmid[⋆†] Marco Canini[‡] Anja Feldmann[†]
[†]TU Berlin [⋆]Aalborg University [‡]KAUST

I. Introduction

Software-defined networks (SDNs) are built on the promise of consistency between control and data plane. Data plane consists of software or hardware and is subject to failures e.g., hardware or software bugs. Sometimes, these failures are outside the knowledge of the control plane consisting of network operating system or the SDN controller. The current state-of-the art tools either check the network statically from the control plane or use tagging or active probe generation to check the data plane behaviour. There is a spectrum of control plane mechanisms [1], [2], [3], [4] which check for some or all of the network-wide invariants like reachability, forwarding loops, waypoint enforcement and slice isolation. While most of the control plane mechanisms cannot model ECMP or NAT, all of them, however, are unaware of the actual behaviour on the data plane and thus, assume that control plane has a consistent view. There is a panoply of data plane mechanisms involving tagging [5], [6] or active probe generation [7], [8] which either check the path or the installed flow rules to monitor the data plane. Moreover, ping and traceroute continue to be the only tools available for network debugging to the network administrators in practice. Bugs in flow table match-action logic or switch hardware can manifest in various ways which are hard to detect and thus, to localize by any of the existing approaches. Moreover, tagging comes with its own limitations. There cannot be available and sufficient space for placing tags for a big scale scenario. Some of the existing mechanisms end up placing a lot of flow rules on the scarce TCAM to take action on tagged packets [5], [9]. The tagging mechanisms which end up tagging a packet on each switch may not detect faults where the output port is same but packet matched a different rule. Considering all of the above limitations, there is a need for a meticulous data plane monitoring tool. In software debugging, we design test cases which check the "code coverage" for finding possible faults, similarly, the monitoring tool should check all of the flow rules and links for possible faults and localize them.

II. Preliminary Analysis

To come up with a rigorous data plane monitoring mechanism, it is important to identify various scenarios which may lead to incorrect behaviour on the data plane. We came up with a scenario where there is a problem in the match part of a specific rule/s in the flow table. This means packets match what they are not supposed to match or do not match what they are supposed to match. We identify these faults or inconsistencies as "match inconsistencies". The problem is undetectable by a tagging mechanism if action of the flow rule matched erroneously is same as the action of expected flow rule. This will result in same path but different flow rule match. Moreover, in OpenFlow, if one adds any non-overlapping rule through the command line utility of the switch e.g., `ovs-ofctl` then there is no notification generated to the controller. Thus, it causes the SDN controller to be unaware of the added flow rule. This is a serious drawback as any adversary can get access to any one switch and insert a flow rule which matches malicious or malformed traffic and thus, compromises the network security. Misconfiguration can also cause such accidental insertion of non-overlapping flow rule.

Key Insights We experimented in an OvS OpenFlow-based soft switches and realised that exisiting mechanisms which use tagging tend to work under strong assumptions. VLAN, MPLS and IP ToS fields are mostly considered for tagging. However, we realise that most of the current networks utilize these fields for forwarding packets and thus, these header fields are not available for tagging. Secondly, tags grow with paths and a limited space field header is not sufficient in a scalable scenario. Thirdly, specific flow rules should be installed in order to take actions on the tagged packets. TCAM is a scarce storage resource and therefore, installing additional flow rules may make TCAM space insufficient for adding further forwarding rules.

On the control plane front, we conducted experiments with the existing control plane mechanisms [1], [2], [3]. HSA [1] and NetPlumber [2] carry out set operations like union, difference, complement and intersection on wildcards representing packet headers. The set operations are efficient on Boolean expression but not on wildcards which make HSA and NetPlumber slow. Binary Decision Diagrams (BDDs) [10] are data structures for boolean expressions and have a much faster support for set operations. APVerifier [3] uses BDD-based header space analysis. An implementation based on APVerifier allows

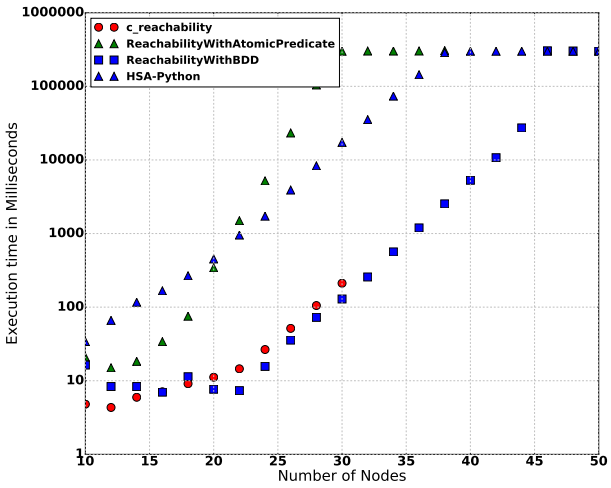

Fig. 1: Execution times of various control plane mechanisms

fast control plane-level calculations to figure out packet header spaces reachable at any point in the topology under consideration. APVerifier has two models one with BDD and the other with Atomic Predicates based on BDD. We carried out our experiments in an ECMP simulation where each node had two links to the next node. At the first node, if the first bit had zero in first bit position, the packet was sent on first link to the second node and if there was one in the first bit, the packet was sent to the other link to second node. On the second node, the second bit of the packet was checked for forwarding in the similar fashion. We continued upto 50 nodes. Figure 1 shows the execution time of HSA, NetPlumber and the two variations of APVerifier. The experiment carried out on OpenFlow-based OpenVSwitches clearly shows that in our ECMP simulation environment, our BDD-based implementation of APVerifier performed better.

III. Research Directions

For an efficient data plane monitoring mechanism, we need to use our key insight to develop a new header field dedicated for tagging. In order to ensure that the data plane monitoring is rigorous, we would like to store *path-level* and *rule-level* information in the tags carried by the new header field. The positioning of the header field is quite tricky as it should be backward compatible with all of the existing protocol headers and in any case, should not interfere in forwarding. Moreover, there should be a support in the switches to take action as per new header field. Since, TCAM is a scarce commodity so its not advisable to install new flow rules specifying actions to be taken on the tagged packet. Therefore, we need to develop set of new actions which can be appended to the existing flow rules to save TCAM space. The tags carrying the rule-level information should include the unique identifiers of the matched flow rules and the tags carrying the path-level should include the unique identifiers of the ports. This information should be sent to the collector. For the collection of tags, filtering with sampling at the edge of the network domain is important to not overload the collector. Similarly, using the BDD-based control plane mechanism, we can calculate the expected tags from the control plane.

Finally, the collector can compare the values obtained from data plane and control plane. If the values match, there is no inconsistency but if they do not match, there is some inconsistency on the data plane. This kind of mechanism will help us to detect most of the rule-based and path-based anomalies.

IV. Summary

Network monitoring is a task of utmost importance as bugs or misconfigurations could seriously impact the network leading to loss of revenues. In order to make sure that the data plane is consistent with the control plane, we need a rigorous data plane verification mechanism that not only checks paths but also the rules matched by the packet on its way. This way we ensure that the network is meticulously covered by the monitoring approach.

Acknowledgement This work was supported by Leibniz Prize project funds of DFG - German Research Foundation: Gottfried Wilhelm Leibniz-Preis 2011 (FKZ FE 570/4-1).

References

[1] P. Kazemian, G. Varghese, and N. McKeown, "Header Space Analysis: Static Checking for Networks," in *NSDI*, 2012.

[2] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. Mckeown, and S. Whyte, "Real Time Network Policy Checking Using Header Space Analysis," in *NSDI*, 2013.

[3] H. Yang and S. S. Lam, "Real-Time Verification of Network Properties Using Atomic Predicates," *IEEE/ACM Transactions on Networking*, vol. 24, pp. 887–900, April 2016.

[4] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE Way to Test Openflow Applications," in *NSDI*, 2012.

[5] S. Narayana, M. Tahmasbi, J. Rexford, and D. Walker, "Compiling Path Queries," in *NSDI*, 2016.

[6] P. Zhang, H. Li, C. Hu, L. Hu, L. Xiong, R. Wang, and Y. Zhang, "Mind the gap: Monitoring the control-data plane consistency in software defined networks," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pp. 19–33, ACM, 2016.

[7] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic Test Packet Generation," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, 2014.

[8] P. Perešíni, M. Kuźniar, and D. Kostić, "Monocle: Dynamic, Fine-Grained Data Plane Monitoring," in *CoNEXT*, 2015.

[9] H. Zhang, C. Lumezanu, J. Rhee, N. Arora, Q. Xu, and G. Jiang, "Enabling Layer 2 Pathlet Tracing through Context Encoding in Software-Defined Networking," in *HotSDN*, 2014.

[10] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 677–691, Aug. 1986.