CS 6250

Final Project Report

Design and implement a real-time subsystem that verifies Loop-Free Property on SDN

Team Members:

Qi Zhou

Lei Zhang

Yongrui Lin

Mengdi Li

Weibin Zhu

1. Abstract

Today's network is complex and prone to bugs. Sometime it's hard for us to detect bugs happened inside the network system, especially loops. From our perspective, it's possible to check network-wide discrepancies in real time, if we build an appropriate network system. The key for our project is to achieve low latency so that we can check the verification time while the network system is not affected. In this project, we use VeriFlow, which is a layer between a software defined networking controller and network devices that checks for network-wide invariant violations dynamically as each forwarding rule is inserted, modified or deleted, to address this problem. We build the network system with Mininet to simulate the controllers and switches to verify our results. Experimental results show that our verification system can successfully detect loops between network switches with an acceptable time.

2. Background

Networks are complex and vulnerable, where detecting and fixing bugs are very important. One of those bugs are the loops that may jeopardize the whole network system and run tons of errors. Therefore, Loop-free property is one of the most important properties that a stable network system should hold. In a traditional network system, the process of verifying the loop-free property is complicated because the distribution routing algorithm is not enough to be applied to examine this property. In a network system, each node maintains its own routing table and can only control itself; however, with a centralized controller in SDN, it is more realistic for us to design and implement a real-time system that verifies the loop-free property before the controller takes any potential buggy action. For our project, we've used this important attribute to design and implement our real time detecting loop-free network system.

3. Verifying Loop-Free Property on SDN

Firstly, the loop-free property is very critical in the real-world networking system as we have already discussed above. Once there exists a loop in the network, which means that there exist some packets that would never be delivered to its final destination as it will iterate in the loop routers forever; or cause the networking system to shut down when severe circumstances applied. From the sender's view, it sends out a packet but it does not get any response; from the receiver's view, it will never receive a complete packet if the packet encounters loops, and it will never know if someone has sent the packet neither. From both sides, it is hard to know what happened inside the network system. If there are too many loops inside a network system, it will crash because no packets will be delivered to its destination. What's worse, even some of the packets are managed to be delivered, the receiver will miss some key pieces of information, that will cause huge trouble. For example, sometimes the users are getting mislead by the wrong information, and the network provider may be dragged into a lawsuit because of this. Consequently, there must be no loop in the network system.

Secondly, Loop-free property can easily be violated in a network system. As we all know that for the physical link, there are so many loops within a network, so as a matter of fact, we are relying on the routing table to make the system function correctly, and to make sure that there are no loops in the network. However, the hard part is that the loop is not decided by one single routing table, it is decided by multiple routing tables. Hence all the routers must coordinate accordingly to ensure that there are no loops in the network system.

Thirdly, SDN makes loop-free property easier to be verified. In a traditional network, all routers are making their own routing table by themselves and there is not a central point that knows the information of all the routers. In the SDN network, we can use Python to implement a central controller which can have all the information of the network. Thus, the SDN can make this

project much easier because we can construct a real-time network system solely depending on our wish.

4. Overview of Approach

We start the project from reading and understanding Veriflow[1], mainly about what Veriflow is and how it works for the verification process. After that, we learned how to simulate the network system we built under the Mininet environment. Secondly, after we got familiar with the Mininet environment, we built a simple version of the subsystem, including implementing controllers that will randomly install/uninstall forward rules, connecting our subsystem with nodes and controllers, and building the simulating networks to test correctness and performance of our subsystem.

Thirdly, we implemented a naive algorithm of the subsystem to detect loops, and ran it under the Mininet environment with randomly manner controllers. We then designed and implemented the verification subsystem that uses an advanced algorithm (Z3 from Microsoft), and run it on Mininet with randomly manner controllers.

Finally, we measured the performance of the advanced verification subsystem that is against the naive verification subsystem. We used Python to construct a network system and we wanted to make it as simple as possible. We used two cases to address this experiment: in the first case, we implemented 4 switches and one controller to detect loops inside the system. In the second case, we simply implemented 3 switches. We tested these two cases to obtain the verification time.

5. Process and Evaluation

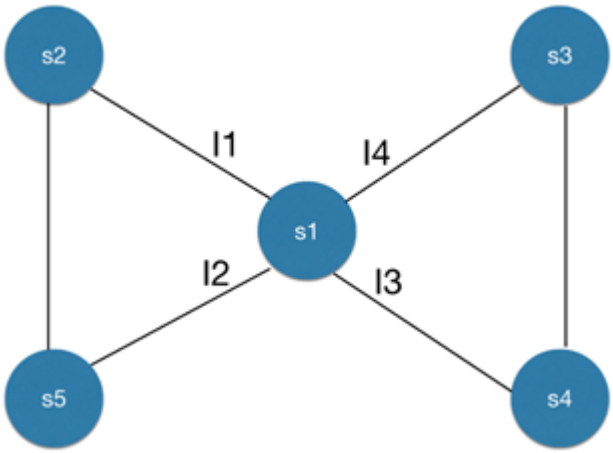

Figure 1

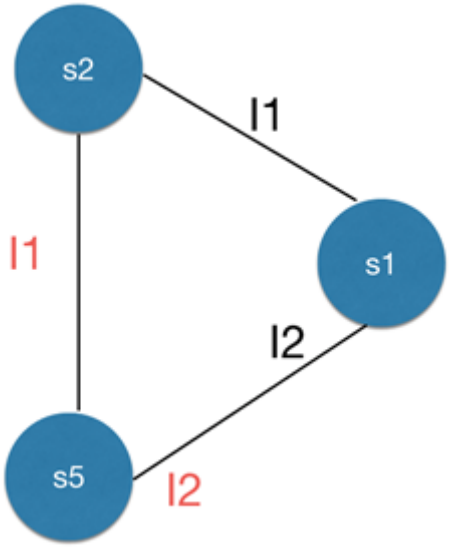

Figure 2

As you can see that in the two figures that are shown above, we are using two examples to demonstrate the algorithm:

In Figure 1, it is a very simple network with five switches. The switches here are the same as routers. As you can see, s1 is connected to s2 by Interface I1, s1 is connected to s5 by Interface I2, as well as connected to s4 by interface I3 and connected to s3 by interface I4. Let's suppose that we add a new route to s1 and on Interface 1(this new route means a set of IP prefixes), when s1 finds a packet and its final destination's IP address is within the set

of IP prefixes that we've added, it will forward the packet to Interface I1 automatically. Therefore, this question becomes if there is a loop in this specific order (the specific order here in Figure2 means that S1 reaches S2 through interface I1, S2 reaches S5 through its own interface I2, and S5 reaches S1 through its own interface I2).

What does the interfaces do? The router does a simple thing: It checks the routing table for the corresponding the IP address, finds the corresponding interface and forwards the packet through that interface. Each interface in a router is corresponding to a set of IP prefixes, and these IP prefixes are 32-bit integers, which can be considered as the lower bound and the upper bound of the IP. Hence, if there exists an IP address which satisfies the lower bound of IP prefix of interface 1 and its upper bound, when S1 gets the packet, it will forward that packet through Interface I1 to S2. Let's suppose that the IP address is still within the range of Interface I1 in S2, then S2 will do the same action, forwarding the packet to S5. Similarly, the same thing will happen to S5, which will send the packet back to S1. Now our question can be simplified as this: is there an IP address or a set of IP addresses that satisfy all these IP prefixes constraints from Interface I1 in S1, Interface I2 in S2 and Interface I2 in S5? This is a well-studied problem, so we can use the standard SMT solver Z3 (provided by Microsoft) to solve this problem. Z3 can tell us whether these IP addresses exist, then we can know whether there are any potential loops inside the system that we've built.

6. Result

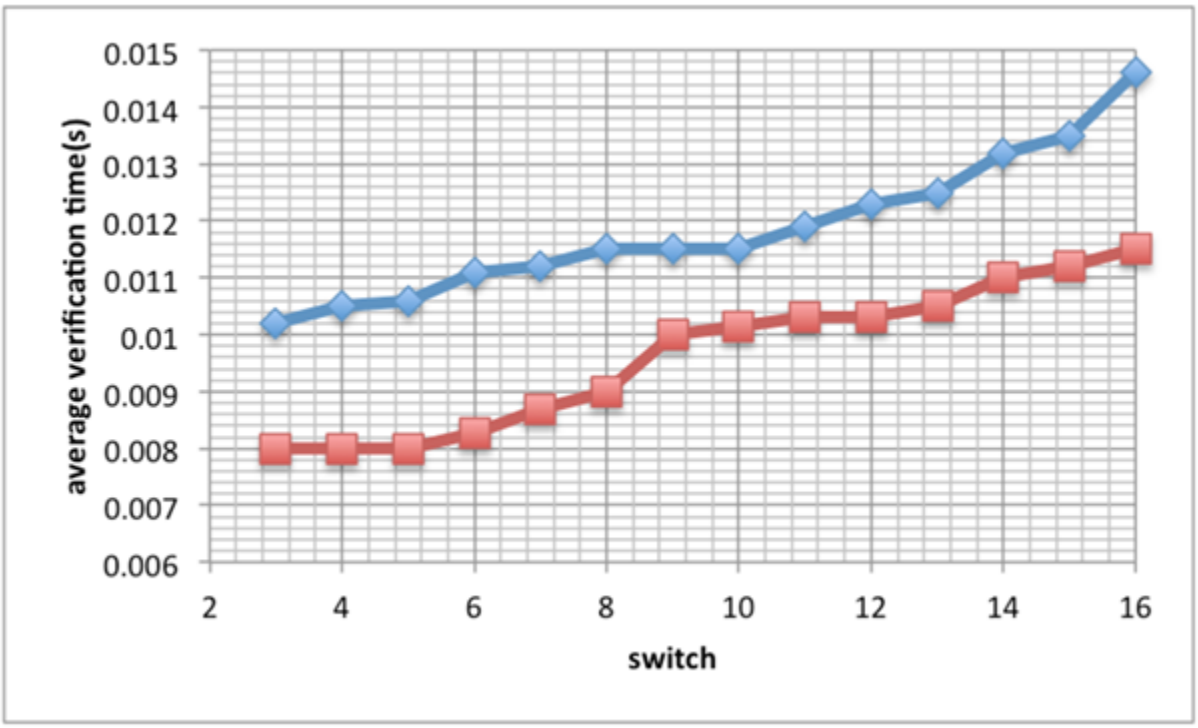

Figure3

As Figure3 shows, the horizontal axis is the number of switches and the vertical axis is the average verification time. What we did was to create the corresponding switches, linked them and make sure that they have the loop-free property before the experiment. And then we ran the experiment and generated 15 different routes to different switches and different interfaces. We tested if any of these loops would violate the loop-free property. The blue line represents the loops that did not violate the loop-free property and the red line represents the loops that violated the loop-free property.

As shown in the graph, the scalability of the network system is decent, because we start from only three routes, and the average intervention time is only about 0.01 second. Increased to 16 different switches, the decision time only increased 50%, to 0.015 seconds. Therefore, when the number of switches increases greatly, the decision time of the switches only increased slightly.

7. Conclusion

During this project, we've successfully implemented a real-time network system to verify the loop-free property in SDN. We used Python to design and implement 5 switches and 1 controller to fit our methods better. Just like the examples shown above, we simplify the approach to our problem a few times, to get the most convincing and pragmatic results. Of course, our approach is just a prototype, there is still a lot of optimization we can do, and we've created a simple and effective model stated above to examine the controllers and swathes.

We've also simulated the system's performance and evaluated its scalability under the Mininet environment. As Figure 3 shows, the loops who didn't not violate the loop-free property consume slightly more time to be verified compared to the loops who violate the loop-free property. However, when the switches increase significantly, the verification time only increase slightly. Therefore, the Loop-Free Property exists in the real-time SDN network system, and it's scalable.

8. References

[1] Khurshid A, Zou X, Zhou W, et al. Veriflow: verifying network-wide invariants in real time[C]//Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013: 15-27.

[2] http://www.veriflow.net/

[3] http://mininet.org/