Source Routing Based In-band Network Telemetry

組員 : 陳泓文 r13922061、葉富銘 r13922146、楊東翰 r13922074、黃恩明 r13922078

Introduction

SDN may lead to the inconsistency faults between the control plane and data plane, and this may get even worse when P4 was introduced, below are the reasons why :

- As a programming language, P4 is prone to bugs inherent and that become potential security threats to the data plane.
- when deploying P4 programs to the hardware, the control-data plane inconsistency possibly appears even the programs are virginal, in the case of network faults occur, including network configuration or hardware failure, operator's misconfiguration, and malicious attacks.

These issues above violate the premise consensus that the logically centralized control plane is consistent with the actual state of the data plane in SDN, which will cause severe damage to the network infrastructure, and lead to unpredictable network security compromise.

Our goal is to quickly discover the control-data plane inconsistency of critical links at runtime in P4-based SDN, e.g., whether the flow rules populated by the controller are strictly executed by the data plane. Given that source routing can specify part of or all nodes that the packet passes through in the end-to-end communication, we can set the critical links as the forwarding paths of probes, thus quickly collecting the state of the data plane at runtime. Therefore, we design a source routing based control-data plane runtime consistency verification mechanism :

1. Generates the active probe traffic with source routing labels,
2. The probe forwards along the source routing path and collects the matching flow rules information.
3. Finally, collected information are compared with the control plane flow rules information through symbolic execution to conduct the consistency verification

Related Work

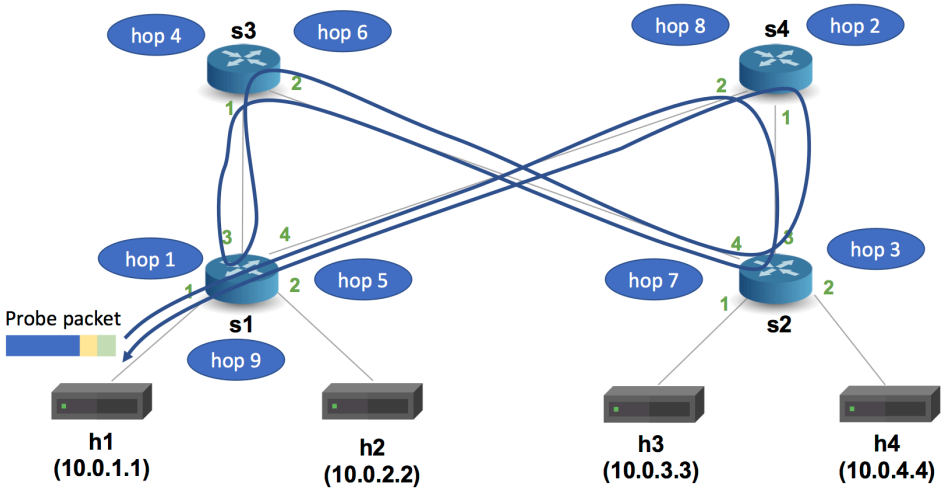
- **Fuzzing-based testing** :
    - P4RL [1] uses fuzzing mechanism based on reinforcement learning to realizes the automatic verification of a single P4 switch at run time,

    - [2] uses fuzzer to find vulnerability a stack buffer overflow in the MPLS parsing code of the OvS slow-path.
    - Time consuming.
- **Symbolic execution-based testing** :
    - P4Consist [3] applies In-band Network Telemetry (INT) [4] to collect the status of the data plane and uses DFS and symbolic execution to implement the control-data plane consistency verification, lots of probing packets needed.
    - SwitchV [5] uses symbolic execution to generate test packet for data plane forwarding behavior and compare the result with BMv2, lots of probing packets needed.
    - p4v [6] converts P4 into GCL language, then use Z3 solver to check the validity
    - Time consuming.

High level design

The design of this system is a source-routing based scheme, in which a packet is sent with a designated path to traverse the switches. During forwarding, a probe is embedded in the packet for further analysis and verification. We can validate that (1) source forwarding is working properly and (2) there is no malfunction of the IPv4 rules in each switch.

The data plane consists of four switches (s1 through s4) and four hosts (h1 through h4), as shown in the following figure:



Each switch is connected to two other switches, forming an “X” topology. The green numbers in the figure indicate the port number of their connection. For instance, switch “s1” connects to switch “s4” via port 4.

To fulfill the aforementioned features: (1) source-routing and (2) inserting probe, the packet format is design as the following tables:

Packet:	ethernet	routing label stack	counter (M)	probe data stack	ipv4
# of bits	112	N * 8	8	M * 40	160

Probe data stack:	bos	switch id	forwarding rule	input port	output port
# of bits	1	8	17	7	7

The *routing label stack* indicates where port the packet should be forwarded so that the N is the number of expected traversed switches. The *counter* records the actual number of switches passed. The initial value of the *counter* is 0, and it will be incremented by 1 when being processed by a switch.

The *probe data stack* is used for recording the *switch index*, *forwarding rule*, *input port* and *output port* in each traversed switch. The *switch index* and *input port* can be used to verify if the source routing is correct so that it follows the exact path given by the user, while *forwarding rule* and *output port* are the values derived from the IPv4 forwarding rule. Place note that the output port is not the port specified in the *routing label stack*.

After the packet is received by the host, the probe stack can be parsed for further validation. For instance, we can lookup the table installed on each switch, checking if the forwarding rule in the probe is matching the one in the table.

Implementation Details

We modify the P4 code based on source routing provided in the official tutorial (p4lang/tutorials). The major modifications are parser and ingress::

- Parser: The parser in P4 can be described by a finite-state machine to extract information from the binary format. Since there is a probe stack located before the IPv4 region, the following code ensures that the probes are skipped to be processed.

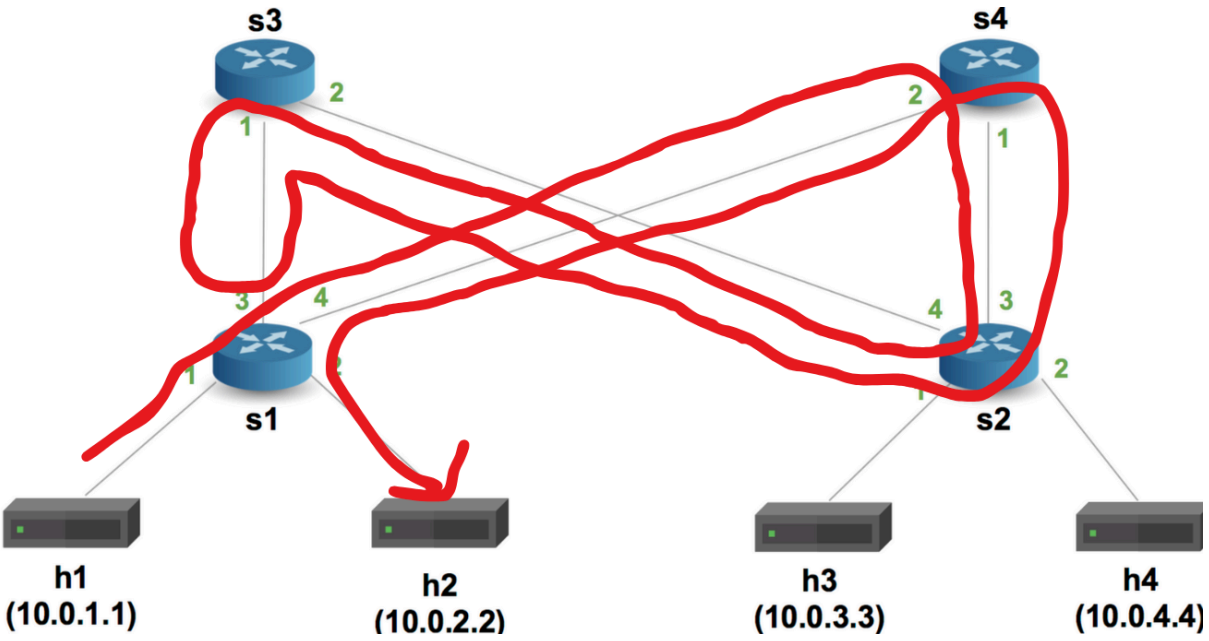
```
state parse_probe_data {
    packet.extract(hdr.probe_data_stack.next);
    transition select(hdr.probe_data_stack.last.bos) {
        1: parse_ipv4;
        default: parse_probe_data;
    }
}
```

- Ingress: The ingress is responsible for determining the target to be forwarded and the output port for the switch to redirect packet. As a result, we insert the probe data into the stack in this stage:`

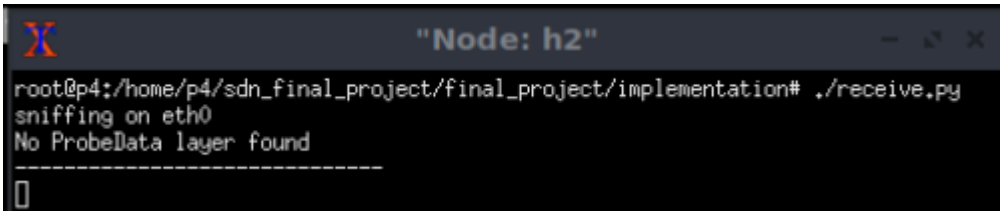
```
hdr.probe_data_stack.push_front(1);
hdr.probe_data_stack[0].setValid();
if (hdr.counter.visited_count == 0) {
    hdr.probe_data_stack[0].bos = 1;
}
else {
    hdr.probe_data_stack[0].bos = 0;
}
swid.apply();
hdr.probe_data_stack[0].rule_id = meta.rule_id;
hdr.probe_data_stack[0].in_port = (bit<7>)standard_metadata.ingress_port;
hdr.probe_data_stack[0].out_port = (bit<7>)standard_metadata.egress_spec;
hdr.counter.visited_count =  hdr.counter.visited_count + 1;

//(3) source routing
if (hdr.routing_label_stack[0].isValid()){
    srcRoute_forward();
    log_msg("878787878");
}else{
    drop();
}
```

  However, due to the lack of information for the switch index, we designed a new table called “swid”. It allows us to install different switch indexes to each switch, enabling obtaining the index of the current switch.

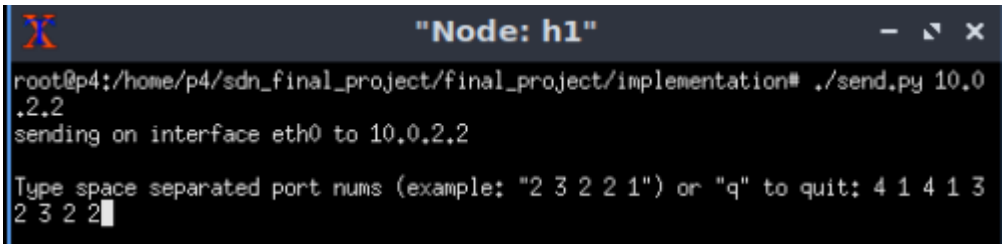
Demo



1. get into the directory: $ cd ./sdn_final_project/final_project/implementation
2. compile the project and run mininet: $ make
3. open the terminal of h1 and h2: mininet> xterm h1 h2
4. start the receive.py in h2: # ./receive.py

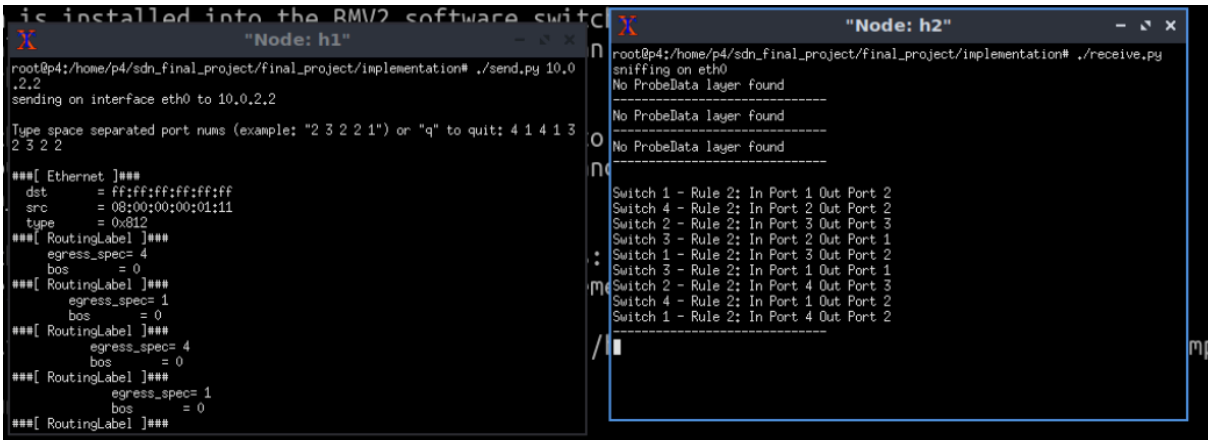


5. start the send.py in h1 and enter the ip of h2 as the destination ip of the 5 tuple in the IPv4 field of the probing packet: # ./send.py 10.0.2.2
6. Then enter the sequence of the source label stack, in this example we enter: 4 1 4 1 3 2 3 2 2



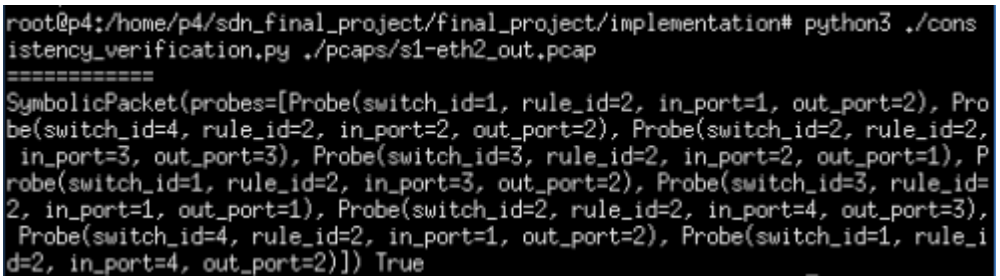
7. Then, we can see the detail of the probing packet that send.py has generated in the terminal of h1. We also see that h2 has received the probing packet and

also print all the info in the probe data stack.



8. Then, we enter ctrl + c in the terminal of h2 to exit.
9. Then we start consistency verification program in h2:

   # python3 ./consistency_verification.py ./pcaps/s1-eth2_out.pcap



10. If you see the result is True, it means that this probing path has pass the verification exam!
11. After all the demo, we exit the mininet and clean all temp file:

    mininet> exit

    $ make stop

    $ make clean

分工表

陳泓文	R13922061	literature collection and review, P4 implementation, Slide, demo recording, demo in report
葉富銘	R13922146	sender implementation, introduction and related work of report
楊東翰	R13922074	Consistency Verification, demo recording
黃恩明	R13922078	Receiver implementation / Report: High-level design & Implementation Details

REFERENCES :

[1] Shukla A, Hudemann K N, Hecker A, et al. Runtime verification of p4 switches with reinforcement learning[C]//Proceedings of the 2019 Workshop on Network Meets AI & ML. 2019: 1-7.

[2] Kashyap T, Shastry B, Fiebig T, et al. Taking control of SDN-based cloud systems via the data plane[C]//Proceedings of the 2018 Symposium on SDN Research. ACM, 2018: 1-13.

[3] Shukla A, Fathalli S, Zinner T, et al. P4Consist: Toward Consistent P4 SDNs[J]. IEEE Journal on Selected Areas in Communications, 2020, 38(7): 1293-1307. doi:10.1109/JSAC.2020.2999653.

[4] In-Band Network Telemetry (INT) Specification V2.1. P4 Language Consortium. Accessed: Mar. 2021. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_ 1.pdf.

[5] Liu J, Hallahan W, Schlesinger C, et al. p4v: Practical Verification for Programmable Data Planes[C]//SIGCOMM 2018. ACM, 2018: 490-503.

[6] Albab K D, DiLorenzo J, Heule S, et al. SwitchV: Automated SDN Switch Validation with P4 Models[C]//SIGCOMM 2022. ACM, 2022: 365-380.