Network Verification Project

Shriya Kaneriya
Advisor : Soudeh Ghorbani

Objective

Implement a network verifier for safety properties, independent of network topology.

Introduction

With the networks becoming more complex today, it is difficult for network administrators to stay on top of the network manually. Correspondingly, there is a need to automate the process through network verifiers, which can aid in designing the network, and maintaining it. Moreover, there is a need for a general verification tool because it is tedious to refer to multiple tools for different types of verification and complex network functions.

We will attempt to verify safety(check that the property stands true eventually) and liveness(check that the property currently stands true) properties within our verifier. This work is currently for the safety property verification, and we will later integrate it with a decomposer for a complete verification tool kit. There are two primary issues that we will attempt to address within verification; namely keeping track of flow tables and performing real time policy checking for network invariants.

We shall be using NetPlumber, a real time policy checking tool. It is an improvement on the Header Space Analysis library. It can verify the state of a network by incrementally incorporating the changes made at the control plane. Thus, to verify safety properties, we can first propose the change on Netplumber at the control plane and verify if the change would still allow all the properties to hold for the new network topology. If so, we can update the network with the proposed changes.

Working with the Library

I spent a lot of time understanding the structure and working of the library [6]. The library is divided into five modules namely Hassel-c, hsa-python, mahak, mininet and netplumber. While the library is a couple of years old, the netplumber module is specifically tagged as out of maintainance. So, I tried to realign the modules and establish the link between them. Since very little documentation exists for the problem, it was difficult to map the starter files and the core code operations of the library.
Each of the modules are written in different programming language for varied purposes and are linked together to run for the NetPlumber application. The first step was to identify the connection between different modules and verify how they function.

Please refer to the code repo at `https://github.com/shriyakaneriya/Network_Verification` for the detailed working as follows.

Header Space Analysis For the header space library, I have outlined the steps to work with modules in the readme file. The researchers provide the stanford dataset in the library, and below I outline the generation of that network. As a setup step to this,
We then try to generate the conversion between different ports in a router. The library also displays the total time it took to compute the possibilities.
Following this, we can check reachability between different source and destination nodes by looping over all the possible source and destination nodes. However, we would like to move a step further and

Fig. 1: Constructing the network using .tf file

```
[skaneri2@masters5 hassel-c]$ ./gen stanford
Parsing: done
Arrays: 43755 (1400160) -> 4189 (134048)
84
2940
57276
110548
128548
144364
189420
228392
248488
267424
281764
294988
306220
316316
379408
417356
682484
Total: 878764 bytes
```

Fig. 2: Checking port forwarding for a router

```
[skaneri2@masters5 hassel-c]$ ./stanford 200002 100003
-> Port: 200002
-> Port: 400006, Rules: bbrb_rtr_12, bbrb_rtr_74, bbrb_rtr_32, _37
-> Port: 100003, Rules: bozb_rtr_29, bozb_rtr_349, bozb_rtr_114, _6
   HS: DX,DX,DX,DX,DX,DX,D172,D20,D0,D4,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 400006, Rules: bbrb_rtr_12, bbrb_rtr_79, bbrb_rtr_32, _37
-> Port: 100003, Rules: bozb_rtr_29, bozb_rtr_349, bozb_rtr_114, _6
   HS: DX,DX,DX,DX,DX,DX,D172,D20,D0,D9,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 400006, Rules: bbrb_rtr_12, bbrb_rtr_100, bbrb_rtr_32, _37
-> Port: 100003, Rules: bozb_rtr_29, bozb_rtr_349, bozb_rtr_114, _6
   HS: DX,DX,DX,DX,DX,DX,D172,D20,D0,D65,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 400006, Rules: bbrb_rtr_12, bbrb_rtr_102, bbrb_rtr_32, _37
-> Port: 100003, Rules: bozb_rtr_29, bozb_rtr_349, bozb_rtr_114, _6
   HS: DX,DX,DX,DX,DX,DX,D172,D20,D0,D67,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 400006, Rules: bbrb_rtr_12, bbrb_rtr_103, bbrb_rtr_32, _37
-> Port: 100003, Rules: bozb_rtr_29, bozb_rtr_349, bozb_rtr_114, _6
   HS: DX,DX,DX,DX,DX,DX,D172,D20,D0,D75,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 1600008, Rules: bbrb_rtr_17, bbrb_rtr_814, bbrb_rtr_32, _41
-> Port: 100003, Rules: yozb_rtr_375, yozb_rtr_545, yozb_rtr_428, _8
   HS: DX,DX,DX,DX,DX,DX,D10,D62,0010xxxx,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 1600008, Rules: bbrb_rtr_17, bbrb_rtr_823, bbrb_rtr_32, _41
-> Port: 100003, Rules: yozb_rtr_375, yozb_rtr_545, yozb_rtr_428, _8
   HS: DX,DX,DX,DX,DX,DX,D10,D63,0010xxxx,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 1600008, Rules: bbrb_rtr_17, bbrb_rtr_834, bbrb_rtr_32, _41
-> Port: 100003, Rules: yozb_rtr_375, yozb_rtr_545, yozb_rtr_428, _8
   HS: DX,DX,DX,DX,DX,DX,D171,D64,0011xxxx,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 1600008, Rules: bbrb_rtr_17, bbrb_rtr_861, bbrb_rtr_32, _41
-> Port: 100003, Rules: yozb_rtr_375, yozb_rtr_545, yozb_rtr_428, _8
   HS: DX,DX,DX,DX,DX,DX,D172,D16,01xxxxxx,DX,DX,DX,DX,D0,D2
_____
-> Port: 200002
-> Port: 1600008, Rules: bbrb_rtr_17, bbrb_rtr_862, bbrb_rtr_32, _41
-> Port: 100003, Rules: yozb_rtr_375, yozb_rtr_545, yozb_rtr_428, _8
   HS: DX,DX,DX,DX,DX,DX,D172,D16,10xxxxxx,DX,DX,DX,DX,D0,D2
_____
Count: 322
Time: 47613 us
```

detect loops (i.e, if a packet can be circulated back to one or more nodes within its path).
Along with that, we would also like to eliminate the possibility that a node is not specifically isolated from all other nodes(i.e. blackholes).
We use the Netplumber for this. We initially began with two possibilities of checking these safety properties. One was to compute all the next possible states and ensure they are safe. However, as networks grow larger, this is usually not possible. So, instead, we would like to verify if the network is safe before every change made to the network. Using a tester code, we can check if a network is loop-free and blackhole-free as we alter a network.
Thus, I demonstrate below two verifier checks on the same network. In the first, I push in a valid topology to indicate that all checks for safety work and the module accepts the changes. In the second I purposefully introduce a loop within the network, and we see that the safety tests fail for the new topology.

Future Steps

- We have to setup a decomposer between liveness and safety property check. The liveness properties can be checked regularly to ensure the network topology is completely active and functioning and the safety check can be run when the

References

[1] Alpern, Bowen, and Fred B. Schneider. ”Recognizing safety and liveness.” Distributed computing 2.3 (1987): 117-126.

Fig. 3: Demonstrating a valid, safe network

```
(netplumber) Shriyas-MacBook-Pro:net_plumbing shriyakaneriya$ python test_net_plumber.py
********************
table: B4
********************
Rule: B4_1 (match: xxx010xx, in_ports = [8], mask: None, rewrite: None, out_ports: [12])
Pipelined To:
Pipelined From:
        B2_1 (101010xx,8 --> 5)
        B2_2 (111010xx,8 --> 5)
Affected By:
Source Flow:
====================
********************
table: B1
********************
Rule: B1_1 (match: 1010xxxx, in_ports = [1], mask: None, rewrite: None, out_ports: [2])
Pipelined To:
Pipelined From:
        client (1010xxxx,1 --> 100)
Affected By:
Source Flow:
  From port 1:
        1010xxxx
====================
Rule: B1_2 (match: 10001xxx, in_ports = [1], mask: None, rewrite: None, out_ports: [2])
Pipelined To:
Pipelined From:
        client (10001xxx,1 --> 100)
Affected By:
Source Flow:
  From port 1:
        10001xxx
====================
Rule: B1_3 (match: 10xxxxxx, in_ports = [1, 2], mask: None, rewrite: None, out_ports: [3])
Pipelined To:
        B3_1 (101xxxxx,3 --> 6)
Pipelined From:
        client (10xxxxxx,1 --> 100)
Affected By:
        B1_1 (1010xxxx,[1])
        B1_2 (10001xxx,[1])
Source Flow:
  From port 1:
        10xxxxxx - (
          1010xxxx U
          10001xxx
          )
====================
====================
********************
table: B2
********************
Rule: B2_1 (match: 1011xxxx, in_ports = [4], mask: D231, rewrite: D8, out_ports: [5
Pipelined To:
        B4_1 (101010xx,5 --> 8)
Pipelined From:
Affected By:
Source Flow:
====================
Rule: B2_2 (match: 10xxxxxx, in_ports = [4], mask: D159, rewrite: D96, out_ports: [5
Pipelined To:
        B4_1 (111010xx,5 --> 8)
Pipelined From:
        B1_1 (1010xxxx,4 --> 2)
        B1_2 (10001xxx,4 --> 2)
Affected By:
        B2_1 (1011xxxx,[4])
Source Flow:
  From port 4:
        11101xxx
  From port 4:
        1110xxxx
====================
********************
table: B3
********************
Rule: B3_1 (match: 101xxxxx, in_ports = [6, 11], mask: D248, rewrite: D7, out_ports
Pipelined To:
Pipelined From:
        B1_3 (101xxxxx,6 --> 3)
Affected By:
Source Flow:
  From port 6:
        101xx111 - 1010x111
====================
********************
source: client
********************
Pipelined To:
        B1_1 (1010xxxx,100 --> 1)
        B1_2 (10001xxx,100 --> 1)
        B1_3 (10xxxxxx,100 --> 1)
Source Flow:
        1xxxxxxx
......checking Source reachability probe
.
----------------------------------------------------------------------
Ran 7 tests in 0.014s

OK
```

Fig. 4: Installing a loop in the network, and rechecking the state to indicate failed safety property test

```
(netplumber) Shriyas-MacBook-Pro:net_plumbing shriyakaneriya$ python test_net_plumber.py
LOOP DETECTED
FFLOOP DETECTED
FFLOOP DETECTED
FFchecking Source reachability probe
LOOP DETECTED
LOOP DETECTED
F
======================================================================
FAIL: testAddRemoveLink (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 189, in testAddRemoveLink
    self.testAddSource()
  File "test_net_plumber.py", line 154, in testAddSource
    self._checkPipelines(pipelines)
  File "test_net_plumber.py", line 110, in _checkPipelines
    self.assertEqual(len(r.next_in_pipeline),pipelines[i][0])
AssertionError: 2 != 1

======================================================================
FAIL: testAddRemoveRule (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 175, in testAddRemoveRule
    self._checkPipelines(pipelines)
  File "test_net_plumber.py", line 110, in _checkPipelines
    self.assertEqual(len(r.next_in_pipeline),pipelines[i][0])
AssertionError: 2 != 0

======================================================================
FAIL: testAddSource (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 154, in testAddSource
    self._checkPipelines(pipelines)
  File "test_net_plumber.py", line 110, in _checkPipelines
    self.assertEqual(len(r.next_in_pipeline),pipelines[i][0])
AssertionError: 2 != 1

======================================================================
FAIL: testRemoveRuleFromPlumbing (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 143, in testRemoveRuleFromPlumbing
    self._checkInfluencedBy(dependencies)
  File "test_net_plumber.py", line 116, in _checkInfluencedBy
    self.assertEqual(len(r.affected_by),dependencies[i])
AssertionError: 0 != 2

======================================================================
FAIL: testRemoveRuleFromPlumbing (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 143, in testRemoveRuleFromPlumbing
    self._checkInfluencedBy(dependencies)
  File "test_net_plumber.py", line 116, in _checkInfluencedBy
    self.assertEqual(len(r.affected_by),dependencies[i])
AssertionError: 0 != 2

======================================================================
FAIL: testRemoveSource (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 159, in testRemoveSource
    self.testAddSource()
  File "test_net_plumber.py", line 154, in testAddSource
    self._checkPipelines(pipelines)
  File "test_net_plumber.py", line 110, in _checkPipelines
    self.assertEqual(len(r.next_in_pipeline),pipelines[i][0])
AssertionError: 2 != 1

======================================================================
FAIL: testSetupPlumbing (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 133, in testSetupPlumbing
    self._checkInfluencedBy(dependencies)
  File "test_net_plumber.py", line 116, in _checkInfluencedBy
    self.assertEqual(len(r.affected_by),dependencies[i])
AssertionError: 0 != 2

======================================================================
FAIL: testSourceReachabilityProbe (__main__.Test)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_net_plumber.py", line 217, in testSourceReachabilityProbe
    self.assertEqual(len(probe_state),2)
AssertionError: 1 != 2

----------------------------------------------------------------------
Ran 7 tests in 0.014s

FAILED (failures=7)
```

[2] Kazemian, Peyman, et al. ”Real time network policy checking using header space analysis.” Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13). 2013.

[3] Kazemian, Peyman, George Varghese, and Nick McKeown. ”Header space analysis: Static checking for networks.” Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[4] Owicki, Susan, and Leslie Lamport. ”Proving liveness properties of concurrent programs.” ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982): 455-495.

[5] Biere, Armin, Cyrille Artho, and Viktor Schuppan. ”Liveness checking as safety checking.” Electronic Notes in Theoretical Computer Science 66.2 (2002): 160-177.

[6] HSA Library: https://bitbucket.org/peymank/hasselpublic/wiki/Home

[7]] NSDI Talk: https://www.usenix.org/conference/nsdi12/technicalsessions/presentation/kazemian