

# SDN Experiment 4

## 实验环境

与第二次实验相同

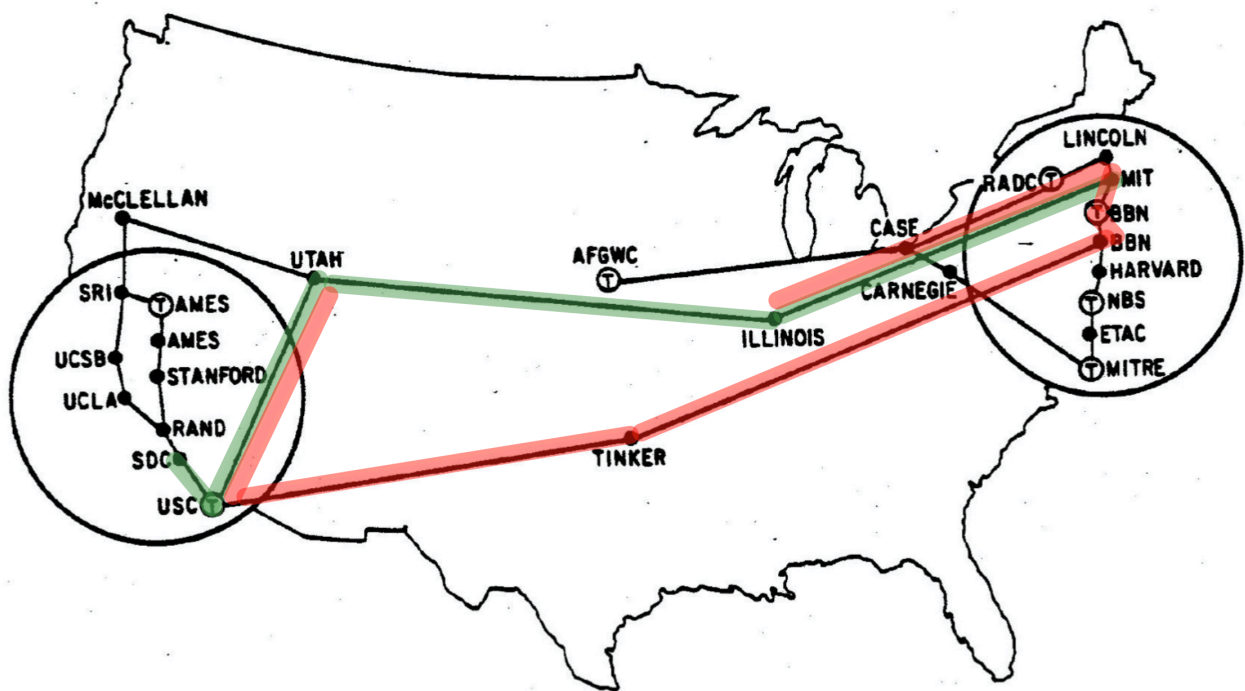
## 实验内容

### 题目

假如你是生活在1972年维护ARPAnet的网络管理员，在前面的实验中你学会了如何建立最短路径，下发了一条SDC到MIT跳数最少的路径（图中绿色的路径）。你的同事Bob某天接到了一个新的需求，要求UTAH到ILLINOIS之间的所有流量必须经过部署于TINKER的流量分析器以进行进一步研究，粗心大意的Bob没有检查当前的网络状态就很快下发了一条新的路径（图中红色的路径）。聪明又机智的你很快意识到Bob下发的流表很可能造成转发的环路。

现要求你运行VeriFlow工具，对上述两条转发路径进行检查，完成下面任务：

1. 打印出环路路径的信息
2. 进一步打印出环路对应的EC的相关信息



### 说明

- 如何观察转发的环路问题？

```
# 1. 启动拓扑
sudo python Arpanet19723.py
```

# 2. 启动最短路径的控制程序

```
ryu-manager ofctl_rest.py shortest_path.py --observe-links
```

# 3. 在拓扑中SDC ping MIT建立连接

```
mininet> SDC ping MIT
```

# 4. Bob下发从UTAH途经TINKER到达ILLINOIS的路径之后，你尝试SDC ping MIT失败

```
python waypoint_path.py
```

# 5. 查看路径上某一个交换机，如USC的流表，发现匹配某一条流表的数据包数目异常增加

# 也可打开wireshark观察该端口，发现不断增加的ICMP Request报文

```
sudo ovs-ofctl dump-flows s22
```

```
onos@zuo-lab:~/sdn-exp/exp4$ ryu-manager shortest_path.py --observe-links
loading app shortest_path.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of NetworkAwareness
creating context network_awareness
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app shortest_path.py of ShortestPath
path: 10.0.0.18 -> 10.0.0.12
10.0.0.18 -> 1:s15:3 -> 3:s22:4 -> 4:s23:2 -> 3:s1:2 -> 2:s25:1 -> 10.0.0.12

*** For testing network connectivity among the hosts, wait a bit for the controller to create all the routes, then do 'pingall' on the mininet console.

*** edited for xjtu sdn_exp_2020

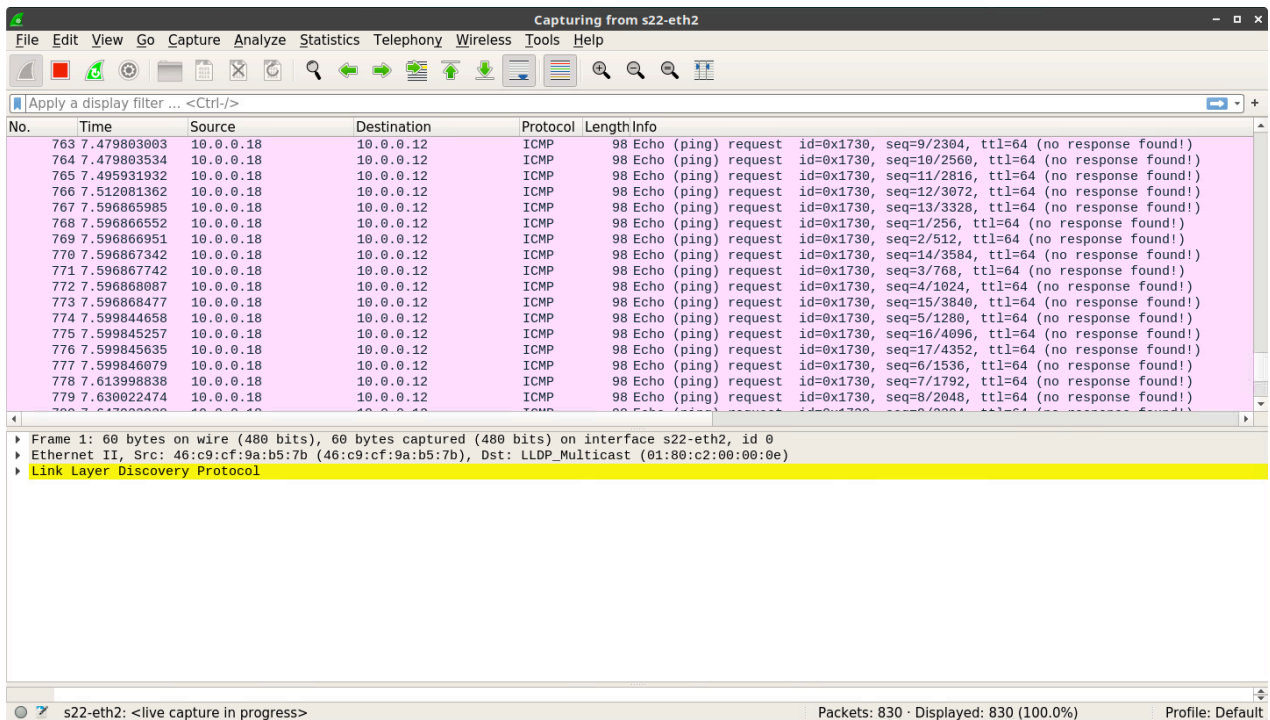
*** Starting CLI:
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=3 ttl=64 time=124 ms
64 bytes from 10.0.0.12: icmp_seq=4 ttl=64 time=230 ms
64 bytes from 10.0.0.12: icmp_seq=5 ttl=64 time=230 ms
64 bytes from 10.0.0.12: icmp_seq=6 ttl=64 time=230 ms
^C
--- 10.0.0.12 ping statistics ---
6 packets transmitted, 4 received, 33% packet loss, time 5047ms
rtt min/avg/max/mdev = 124.888/203.957/230.585/45.653 ms
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
^C
--- 10.0.0.12 ping statistics ---
17 packets transmitted, 0 received, 100% packet loss, time 16373ms

mininet> []

onos@zuo-lab:~/sdn-exp/exp4$ sudo python waypoint_path.py
install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1

onos@zuo-lab:~/sdn-exp/exp4$ []

s22-eth4"
cookie=0x0, duration=40.755s, table=0, n_packets=0, n_bytes=0, priority=1,ip,in_port="s22-eth4",nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=output:"s22-eth2"
cookie=0x0, duration=40.740s, table=0, n_packets=1198, n_bytes=117404, priority=1,ip,in_port="s22-eth2",nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=output:"s22-eth4"
cookie=0x0, duration=182.029s, table=0, n_packets=30, n_bytes=2675, priority=0 actions=CONTROLLER:65535
onos@zuo-lab:~/sdn-exp/exp4$ sudo ovs-ofctl dump-flows s22
cookie=0x0, duration=182.795s, table=0, n_packets=66, n_bytes=3960, priority=65535,dl_dst=01:80:c2:00:00:0e,dl_type=0x88cc actions=CONTROLLER:65535
cookie=0x0, duration=144.189s, table=0, n_packets=20, n_bytes=1960, priority=1,ip,in_port="s22-eth3",nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=output:"s22-eth4"
cookie=0x0, duration=41.546s, table=0, n_packets=0, n_bytes=0, priority=1,ip,in_port="s22-eth4",nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=output:"s22-eth2"
cookie=0x0, duration=41.531s, table=0, n_packets=1300, n_bytes=127400, priority=1,ip,in_port="s22-eth2",nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24 actions=output:"s22-eth4"
cookie=0x0, duration=182.820s, table=0, n_packets=30, n_bytes=2675, priority=0 actions=CONTROLLER:65535
onos@zuo-lab:~/sdn-exp/exp4$
```



- 最短路径算法中为何使用rest api下发表？

由于VeriFlow仅支持OpenFlow1.0, `shortest_path.py` 与 `waypoint_path.py` 中使用rest api 更简便。rest api所用到的文件 `ofctl_rest.py` 位于路径`ryu/ryu/app/`中

- 如何使用VeriFlow

```
# 0. 从github下载VeriFlow并打上实验补丁
git clone https://github.com/samueljero/BEADS.git
cd BEADS
git am 0001-for-xjtu-sdn-exp-2020.patch

# 1. 编译VeriFlow
cd veriflow/VeriFlow
make clean all

# 2. 在自定义端口开启远程控制器，运行最短路程序
ryu-manager ofctl_rest.py shortest_path.py --ofp-tcp-listen-port 1024 --
observe-links

# 3. 运行VeriFlow的proxy模式
VeriFlow的proxy模式的cmd格式为：
VeriFlow <veriflow_port> <controller_address> <controller_port> <topology_file>
<log_file>
可用如下命令运行VeriFlow的proxy模式：
./VeriFlow 6633 127.0.0.1 1024 Arpanet19723.txt log_file.txt

# 4. 启动拓扑
sudo python Arpanet19723.py

# 5. 在拓扑中SDC ping MIT建立连接
mininet> SDC ping MIT
```

# 6. 下发从UTAH途经TINKER到达ILLINOIS的路径, 在log文件中观察VeriFlow检测到的环路信息  
python waypoint\_path.py

```
Terminal - test@sdnexp: ~/Desktop/sdn/mininet/custom
*** Type 'exit' or control-C to shut down network
*** For testing network connectivity among the hosts, wait a bit for the controller to create all the routes, then do 'pingall' on the mininet console.
*** edited for xjtu_sdn_exp_2020

*** Starting CLI:
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data:
64 bytes from 10.0.0.12: icmp seq=2 ttl=64 time=258 ms
64 bytes from 10.0.0.12: icmp seq=3 ttl=64 time=254 ms
64 bytes from 10.0.0.12: icmp seq=4 ttl=64 time=231 ms
64 bytes from 10.0.0.12: icmp seq=5 ttl=64 time=232 ms
^C
--- 10.0.0.12 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4028ms
rtt min/avg/max/mdev = 231.818/244.482/258.867/12.439 ms
mininet>

Terminal - test@sdnexp: ~/sdn/VeriFlow
test.o net.o thread.o StringTokenizer.o -lpthread
test@sdnexp:~/sdn/VeriFlow$ ./VeriFlow 6633 127.0.0.1 1998 ../exp-4/Arpanet19723.txt ../exp-4/log
id 125 ipAddress 10.0.0.25 endDevice 1 port 0 nextHopIpAddress 20.0.0.23
id 122 ipAddress 10.0.0.22 endDevice 1 port 0 nextHopIpAddress 20.0.0.20
id 120 ipAddress 10.0.0.20 endDevice 1 port 0 nextHopIpAddress 20.0.0.21
id 117 ipAddress 10.0.0.17 endDevice 1 port 0 nextHopIpAddress 20.0.0.13
id 116 ipAddress 10.0.0.16 endDevice 1 port 0 nextHopIpAddress 20.0.0.11
id 115 ipAddress 10.0.0.15 endDevice 1 port 0 nextHopIpAddress 20.0.0.8
id 118 ipAddress 10.0.0.18 endDevice 1 port 0 nextHopIpAddress 20.0.0.15
id 114 ipAddress 10.0.0.14 endDevice 1 port 0 nextHopIpAddress 20.0.0.12
id 113 ipAddress 10.0.0.13 endDevice 1 port 0 nextHopIpAddress 20.0.0.2

Terminal - test@sdnexp: ~/Desktop/sdn/ryu/ryu/app
test@sdnexp:~/Desktop/sdn/ryu/ryu/app$ ryu-manager ofctl_rest.py shortest_path.py --of-tcp-listen-port 1998 --observe-links
loading app ofctl_rest.py
loading app shortest_path.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app None of NetworkAwareness
creating context network_awareness
instantiating app shortest_path.py of ShortestPath
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ofctl_rest.py of RestStatsApi
(30696) wsgi starting up on http://0.0.0.0:8080
path: 10.0.0.18 -> 10.0.0.12
10.0.0.18 -> 1:s15:3 -> 3:s22:4 -> 4:s23:2 -> 3:s1:2 -> 2:s25:1 -> 10.0.0.12

Terminal - test@sdnexp: ~/Desktop/sdn/ryu/ryu/app
test@sdnexp:~/Desktop/sdn/ryu/ryu/app$ sudo python waypoint_path.py
<Response [200]>
<Response [200]>
install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1
test@sdnexp:~/Desktop/sdn/ryu/ryu/app$ sudo python waypoint_path.py
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
<Response [200]>
install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1
test@sdnexp:~/Desktop/sdn/ryu/ryu/app$
```

## ● VeriFlow的主要类或函数

# 1. VeriFlow::main()

VeriFlow的程序入口, 规定了test模式和proxy模式的调用格式

# 2. VeriFlow::parseTopologyFile()

VeriFlow解析拓扑文件, 建立网络模型的函数, 规定了拓扑文件的格式

# 3. VeriFlow::handleVeriFlowConnection()

处理socket连接关系的函数, 每个连接拥有两个单向通信线程, 实现控制器和交换机之间的双向通信

# 4. OpenFlowProtocolMessage::process()

处理OpenFlow消息的入口函数, 根据消息的类型调用相应的处理函数

# 5. OpenFlowProtocolMessage::processFlowRemoved()

处理OFPT\_FLOW\_REMOVED消息的函数

# 6. OpenFlowProtocolMessage::processFlowMod()

处理OFPT\_FLOW\_MOD消息的函数

# 7. EquivalenceClass

表示VeriFlow定义的等价类的数据结构, 包括每个域的名称和存储的顺序

# 8. VeriFlow::verifyRule()

执行VeriFlow核心算法的函数, 包括对等价类的划分、转发图的构造与不变量的验证

# 9. VeriFlow::traverseForwardingGraph()

## 示例

- 环路路径的打印 本实验要求打印出环路的信息，包括出现环路的提示信息，EC的基本信息和环路路径上的IP地址 提示：traverseForwardingGraph函数中的visited为unordered\_set，可改成有序的数据结构

```
File Edit Search View Document Help
/home/test/Desktop/sdn/exp-4/log - Mousepad
599
600 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
601 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583
602
603 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
604 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (0-00:00:00:00
605 [VeriFlow::traverseForwardingGraph] Loop path is:
606 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
607
608 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
609 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582
610 [VeriFlow::traverseForwardingGraph] Loop path is:
611 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
612
613 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
614 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583
615 [VeriFlow::traverseForwardingGraph] Loop path is:
616 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
617
618 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
619 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (0-00:00:00:00
620 [VeriFlow::traverseForwardingGraph] Loop path is:
621 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
622
623 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
624 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582
625 [VeriFlow::traverseForwardingGraph] Loop path is:
626 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
627
628 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
629 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583
630 [VeriFlow::traverseForwardingGraph] Loop path is:
631 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
632
633
```

- 相关数据包信息的打印 EC的基本信息显示为14个域的区间形式，为方便Bob查错，现简化EC信息的表示形式，仅从14个域中提取TCP/IP五元组作为主要信息显示 提示：在环路路径打印的基础上，修改EC的显示格式

```
File Edit Search View Document Help
/home/test/Desktop/sdn/exp-4/log - Mousepad
6005
6006 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
6007 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6008
6009 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
6010 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6011
6012 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
6013 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6014
6015 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6016 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6017 [VeriFlow::traverseForwardingGraph] Loop path is:
6018 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
6019
6020 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6021 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6022 [VeriFlow::traverseForwardingGraph] Loop path is:
6023 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
6024
6025 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6026 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6027 [VeriFlow::traverseForwardingGraph] Loop path is:
6028 20.0.0.25 -> 20.0.0.1 -> 20.0.0.23 -> 20.0.0.22 -> 20.0.0.9 -> 20.0.0.16 -> 20.0.0.7 -> 20.0.0.25
6029
6030 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6031 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6032 [VeriFlow::traverseForwardingGraph] Loop path is:
6033 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
6034
6035 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6036 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6037 [VeriFlow::traverseForwardingGraph] Loop path is:
6038 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
6039
6040 [VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
6041 [VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0-10.0.0.255), nw_dst (10.0.0.0-10.0.0.255), nw_proto(0-255), tp_src(0-65535), tp_dst(0-65535)
6042 [VeriFlow::traverseForwardingGraph] Loop path is:
6043 20.0.0.25 -> 20.0.0.7 -> 20.0.0.16 -> 20.0.0.9 -> 20.0.0.22 -> 20.0.0.23 -> 20.0.0.1 -> 20.0.0.25
```

## 参考

- 第二次实验参考自李呈的 [network awareness](#) 项目
- VeriFlow 的使用说明参考 [README](#)
- VeriFlow 相关论文，汇报视频请参考[NSDI'13会议网站](#)