

实验四指导书

- 实验目的
- 问题背景
- VeriFlow使用说明
- 基础实验部分
 - 结果示例
- 拓展实验部分
 - 运行示例
- 参考

实验目的

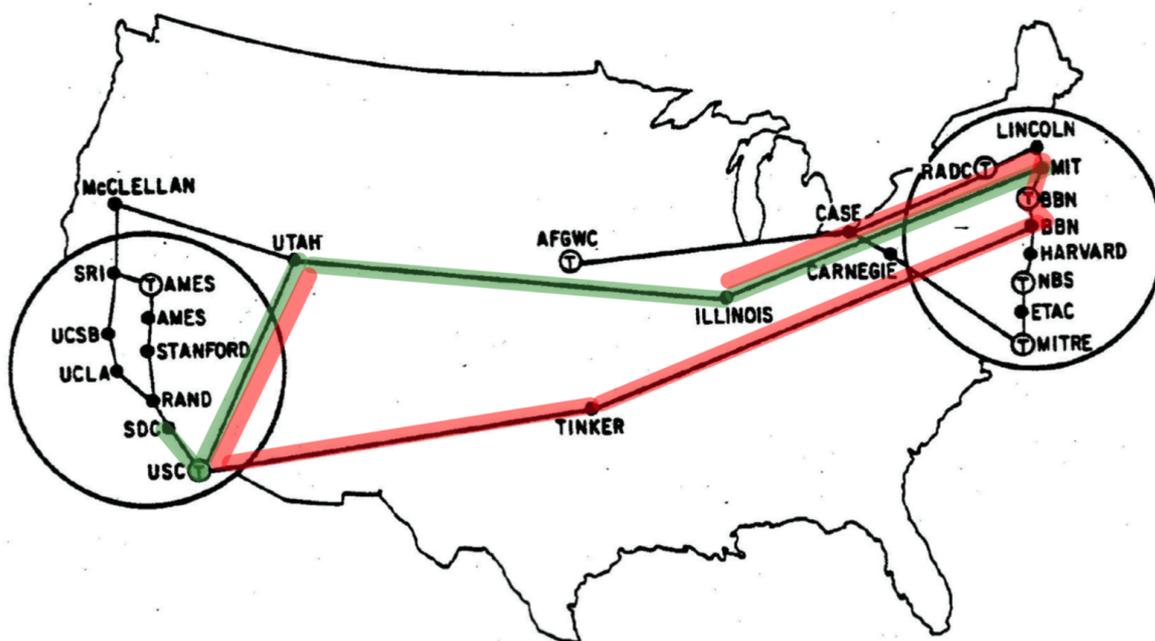
通过本次实验，希望大家掌握以下内容：

- 理解网络故障的普遍性及其造成的严重后果。
- 学习网络验证工具 VeriFlow 的工作原理。
- 掌握在 SDN 网络中使用 VeriFlow 检测网络故障的方法。

问题背景

在第三次实验中，你下发了一条 UTAH 到 ILLINOIS 并且途径 TINKER 的路径，即下图中的红色路径。另一位网络管理员 Bob，没有仔细检查就下发了一条从 SDC 到 MIT 跳数最少的路径，即下图中的绿色路径。

你很快意识到，Bob 下发的路径可能造成转发环路。现要求你运用 VeriFlow 工具，对上述两条转发路径进行检查，完成后续实验任务。



VeriFlow使用说明

- 如何观察转发环路？

- # 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
sudo 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

```

~/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 -> 1s15:3 -> 3s22:4 -> 4s23:2 -> 3s1:2 -> 2s25:1 -> 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> []

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

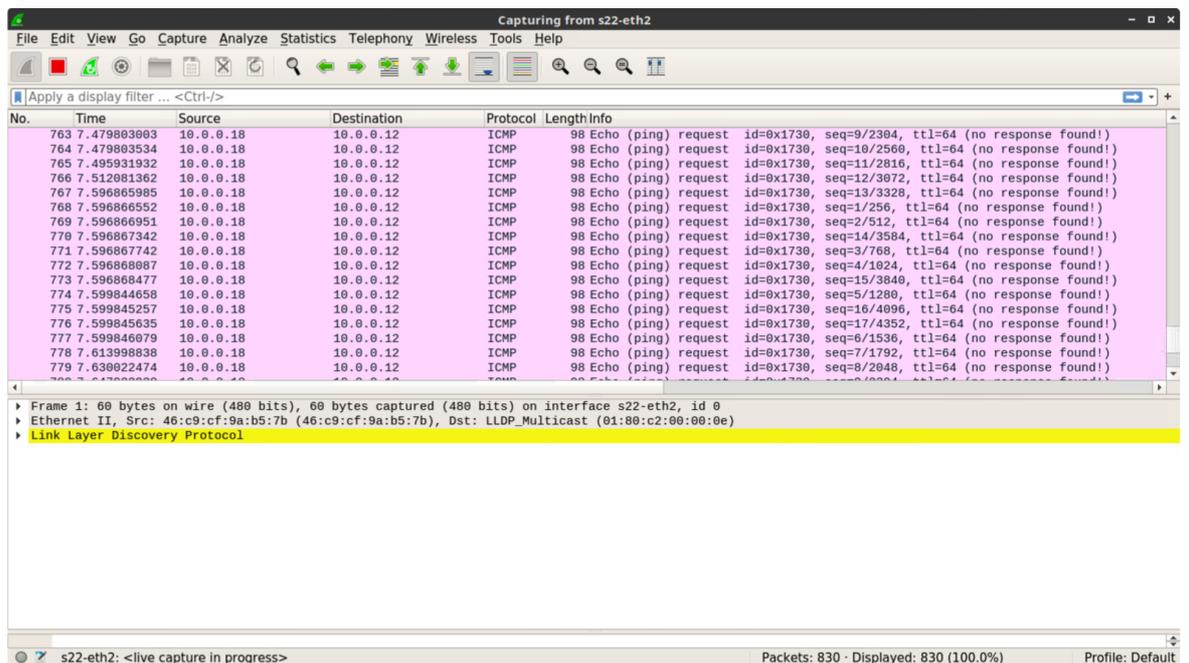
~/sdn-exp/exp4$ []

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

~/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=0x80cc 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

~/sdn-exp/exp4$

```



- 最短路径算法中为何使用rest api下发流表?

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
(Arpanet19723.txt提前放在VeriFlow.o同一文件夹下)

# 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
sudo python waypoint_path.py
```

The image contains four terminal screenshots arranged in a 2x2 grid. The top-left terminal shows the execution of `ryu-manager` and a successful `SDC ping MIT` test. The top-right terminal shows the execution of `ryu-manager` with `shortest_path.py` and `observe-links` options, displaying the loading of various modules and the instantiation of the `ShortestPath` app. The bottom-left terminal shows the execution of `./VeriFlow` with specific parameters, displaying a list of network nodes and their IP addresses. The bottom-right terminal shows the execution of `waypoint_path.py`, displaying the installation of a waypoint path and the resulting network topology.

- VeriFlow源码中的主要类或函数

```
# 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()

遍历某个特定EC的转发图，验证是否存在环路或黑洞

基础实验部分

1. 输出每次影响EC（等价类）的数量
2. 打印出环路路径的信息
3. 进一步打印出环路对应的EC的相关信息
4. 分析原始代码与补丁代码的区别，思考为何需要添加补丁

结果示例

- EC数目的打印

对每条验证的规则，实验要求输出这条规则所影响的EC数目

```
File Edit Search View Document Help
/home/test/Desktop/sdn/exp-4/log - Mousepad
684 [VeriFlow::traverseForwardingGraph] Found a BACK POSE for the following packet class as current location (20.0.0.25) not found in the graph.
685 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583-01:80:c2:00:00:0f, 28
686
687 [VeriFlow::verifyRule] verifying this rule: [Rule] type: 1, dSrcAddr: 00:00:00:00:00, dSrcAddrMask: 0:0:0:0:0, dDstAddr: 00:00:00:00:00, dDstAddrMask: 0:0:0:0:0,
688 [VeriFlow::verifyRule] eccount: 3
689
690 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
691 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (0-00:00:00:00:00, 165252221581-
692
693 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
694 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582-01:80:c2:00:00:0e, 16
695
696 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
697 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583-01:80:c2:00:00:0f, 28
698
699 [VeriFlow::verifyRule] verifying this rule: [Rule] type: 1, dSrcAddr: 00:00:00:00:00, dSrcAddrMask: 0:0:0:0:0, dDstAddr: 00:00:00:00:00, dDstAddrMask: 0:0:0:0:0,
700 [VeriFlow::verifyRule] eccount: 3
701
702 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
703 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (0-00:00:00:00:00:00, 165252221581-
704
705 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
706 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582-01:80:c2:00:00:0e, 16
707
708 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
709 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583-01:80:c2:00:00:0f, 28
710
711 [VeriFlow::verifyRule] verifying this rule: [Rule] type: 1, dSrcAddr: 00:00:00:00:00, dSrcAddrMask: 0:0:0:0:0, dDstAddr: 00:00:00:00:00, dDstAddrMask: 0:0:0:0:0,
712 [VeriFlow::verifyRule] eccount: 3
713
714 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
715 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (0-00:00:00:00:00:00, 165252221581-
716
717 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
718 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582-01:80:c2:00:00:0e, 16
719
720 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.15.
721 [VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583-01:80:c2:00:00:0f, 28
```

- 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 -> 20.0.0.22 -> 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
```

- 相关数据包信息的打印

EC的基本信息显示为14个域的时间形式, 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
6006 [VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
6007 [VeriFlow::traverseForwardingGraph] PacketClass: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), 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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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: mw_src (10.0.0.0-10.0.0.255), mw_dst (10.0.0.0-10.0.0.255), mw_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
```

- 分析原始代码与补丁代码的区别, 思考为何需要添加补丁

添加完补丁之后, 可用以下命令查看补丁修改的文件内容, 按q退出。(提示: 可以先运行未打补丁的VeriFlow代码, 将其输出结果与实验中的结果进行比较, 再结合 `git diff` 显示的代码修改内容, 得出打补丁的原因)

```
git diff HEAD origin/HEAD
```

拓展实验部分

1. VeriFlow 的验证速度仍然不能匹配网络本身的实时性要求, 你可以尝试对 VeriFlow 作出改进或提出新的验证方法, 提升验证速度。
2. VeriFlow 的官方源码尚未公开, 本指导书找到的 VeriFlow 代码实际上存在一些细节问题 (不影响基础实验部分的完成), 你可以尝试找出并解决这些问题。

运行示例

- 数据集

采用 Internet2 数据集测量 VeriFlow 的性能，该数据集共有9个交换机，每个交换机上大约8000条转发规则，拓扑见 `Internet2_topo.py`。

- 测量一条规则的运行时间

`VeriFlow.cpp` 下的 `verifyRule()` 函数，用来验证一条规则是否会引起环路、黑洞等错误。其中使用 `updateTime`、`packetClassSearchTime`、`graphBuildTime`、`queryTime` 四个变量，分别记录 VeriFlow 在验证一条规则过程中四个阶段的用时，单位是 **us**。四个时间相加就是验证一条规则的总用时。

- 运行步骤（拓展任务一）

假设之前的内容你都已经完成，继续参考以下步骤：

```
# 1. 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

# 2. 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 Internet2_topo.txt log_file.txt
(Internet2_topo.txt提前放在VeriFlow.o同一文件夹下)

# 3. 启动拓扑
sudo python Internet2_topo.py

# 4. 下发规则
mininet> sudo python i2_rule.py

# 5. 观察统计用时
```

- 代码本身的问题举例（拓展任务二）

例如 `veriflow.cpp` 中使用变量 `faults` 统计出错的等价类数量，但实际代码中对 `faults` 的更新是有问题的，你可能会观察到 `faults` 逐渐增大到一个不正确的数值。你可以根据自己的理解修改对 `faults` 的更新机制，也可以自行探索并解决其他细节问题（更鼓励）。

参考

- VeriFlow工具的使用说明参考[VeriFlow开源项目](#)
- VeriFlow相关论文，汇报视频请参考[NSDI'13会议网站](#)
- 更多与打补丁相关的命令请参考[git-am](#)
- 网络验证相关论文 `delta-net`，可以参考[论文网站](#)