实验四指导书

实验目的

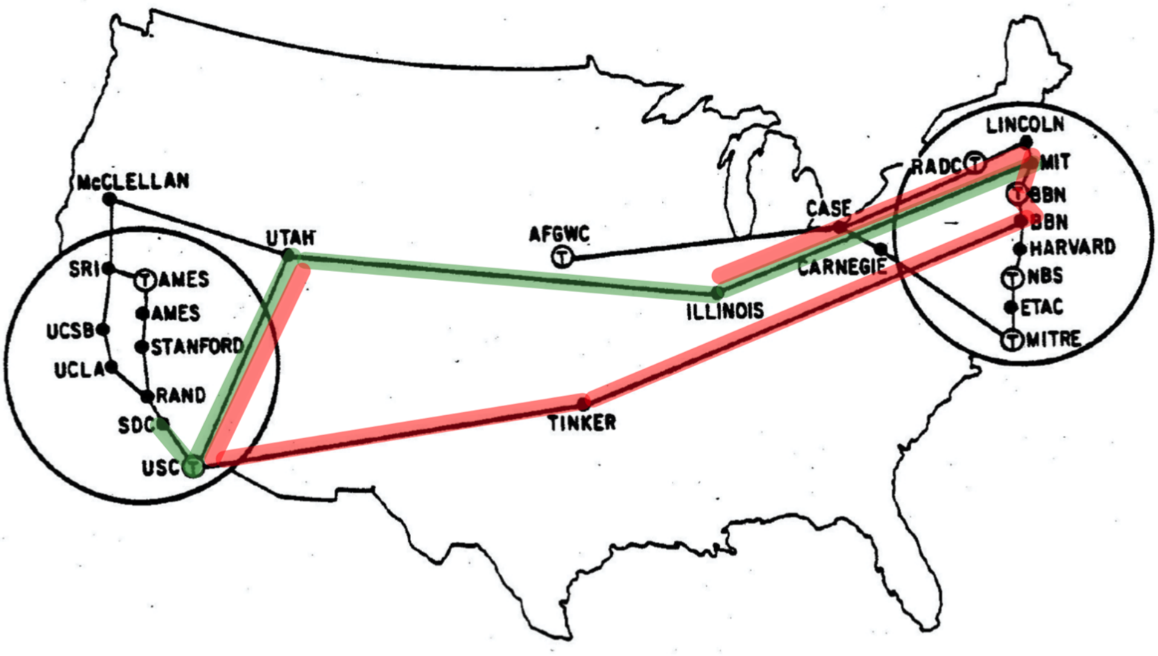
通过本次实验，希望大家掌握以下内容：

- 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。

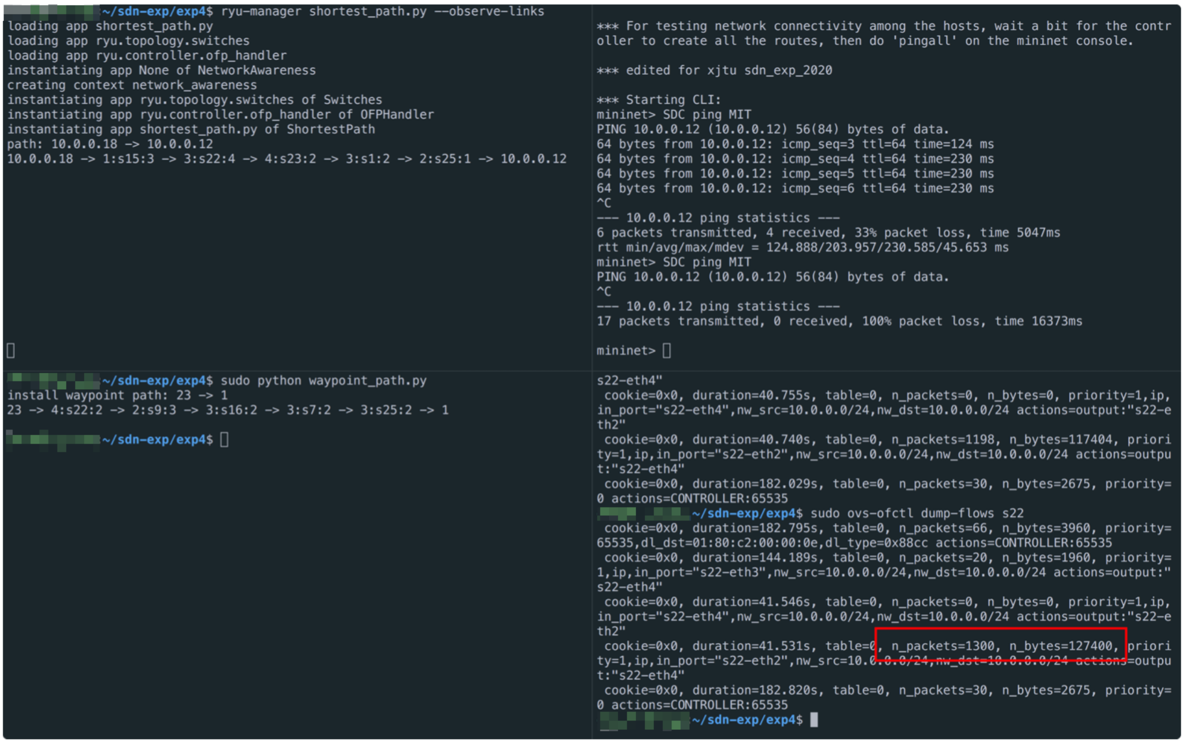
你很快意识到，`Bob` 下发的路径可能造成转发环路。现要求你运用 `veriFlow` 工具，对上述两条转发路径进行检查，完成后续实验任务。

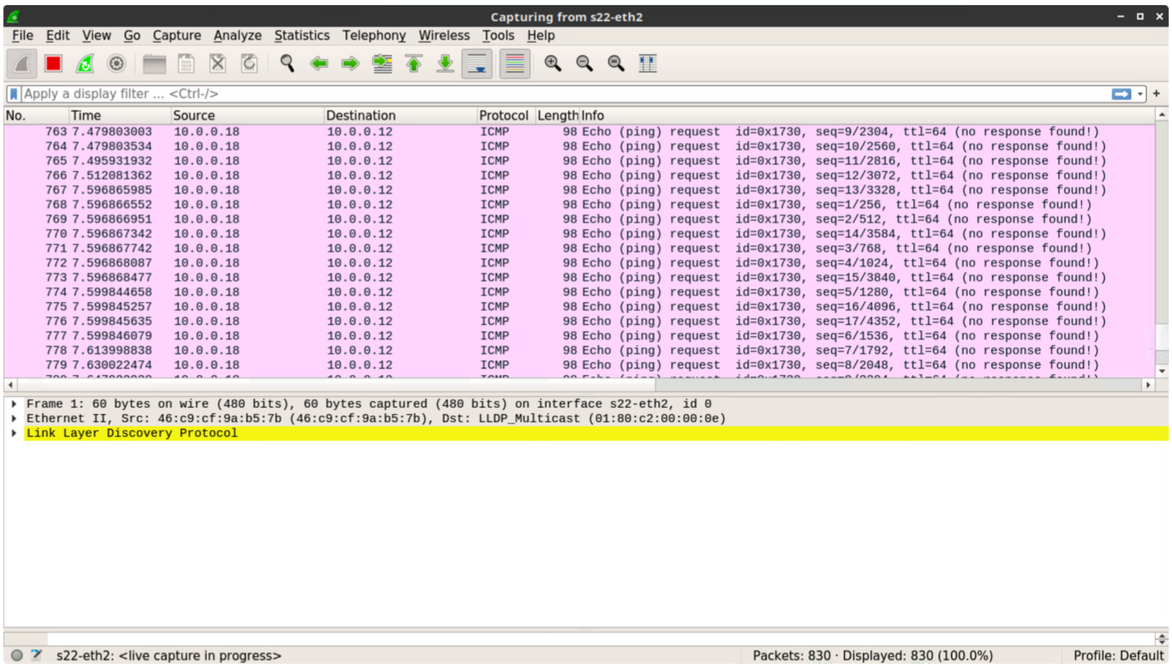


VeriFlow使用说明

- 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
```



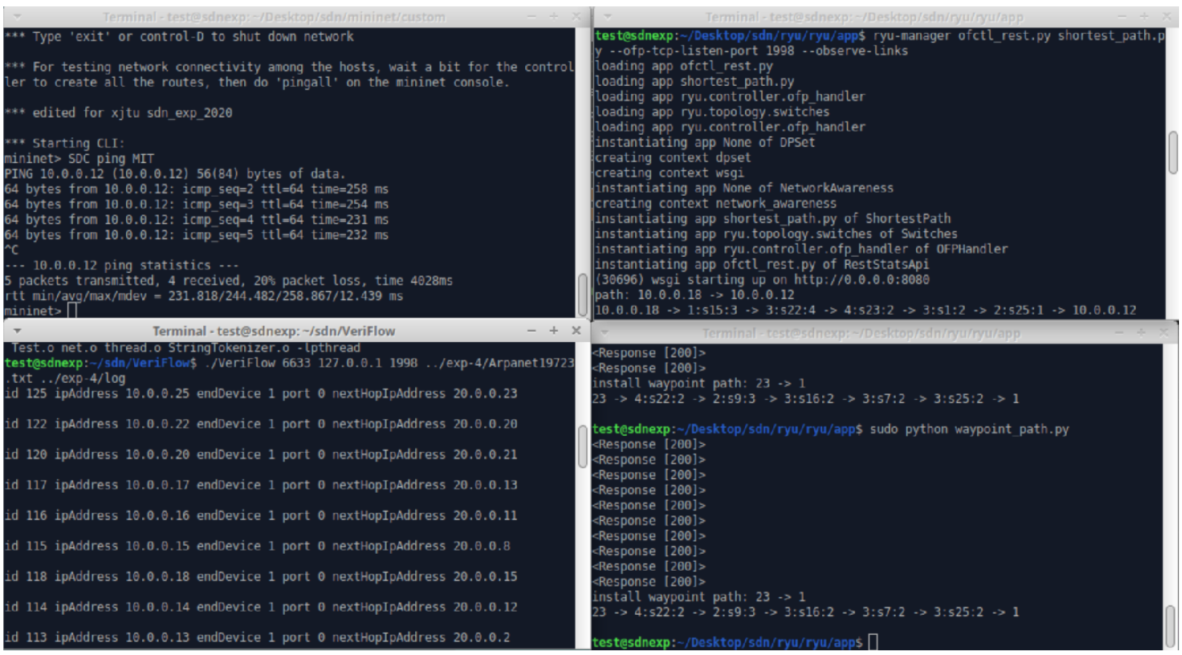


- 最短路径算法中为何使用rest api下发流表?

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
```



- VeriFlow源码中的主要类或函数

```
# 1. VeriFlow::main()
```

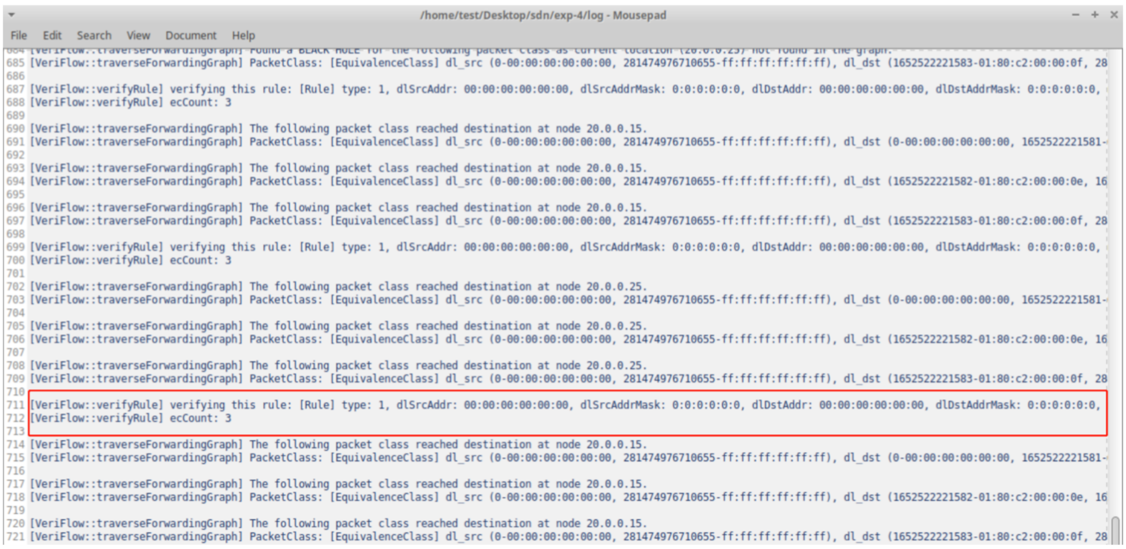
```
VeriFlow的程序入口，规定了test模式和proxy模式的调用格式

# 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

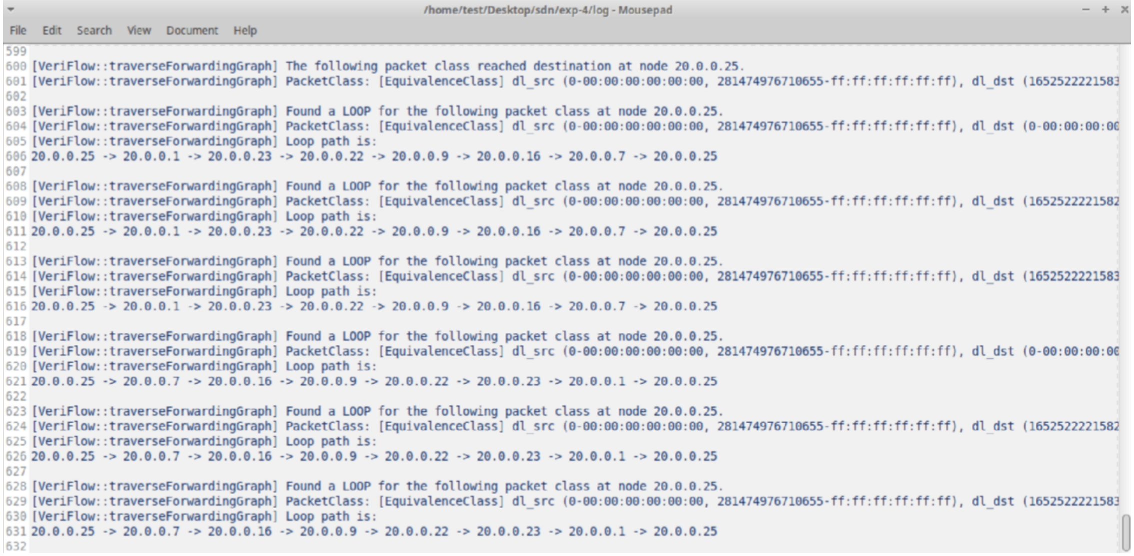
结果示例

- 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



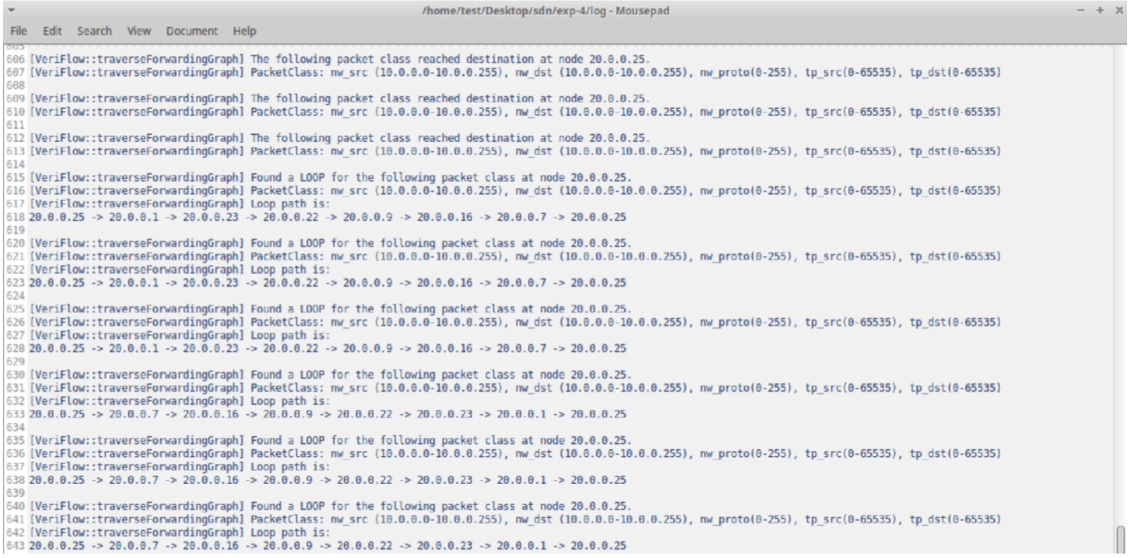
- 环路路径的打印
  本实验要求打印出环路的信息，包括出现环路的提示信息，EC的基本信息和环路路径上的IP地址

提示：traverseForwardingGraph函数中的visited为unordered_set，可改成有序的数据结构



```
/home/test/Desktop/sdn/exp-4/log - Mousepad
File Edit Search View Document Help

[VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (1652522221583

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (1652522221582
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (1652522221583
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (1652522221582
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (1652522221583
[VeriFlow::traverseForwardingGraph] Loop path is:
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
```

- 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



```
/home/test/Desktop/sdn/exp-4/log - Mousepad
File Edit Search View Document Help

[VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
[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)

[VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
[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)

[VeriFlow::traverseForwardingGraph] The following packet class reached destination at node 20.0.0.25.
[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)

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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

[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node 20.0.0.25.
[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)
[VeriFlow::traverseForwardingGraph] Loop path is:
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