# SDN第四次实验

目录

[SDN第四次实验 1](#_Toc134285728)

[实验要求 1](#_Toc134285729)

[1.基础实验部分 1](#_Toc134285730)

[2.拓展实验部分 1](#_Toc134285731)

[准备部分 1](#_Toc134285732)

[安装veriFlow并编译 1](#_Toc134285733)

[在自定义端口开启远程控制器，运行最短路程序 1](#_Toc134285734)

[运行VeriFlow的proxy模式 2](#_Toc134285735)

[启动拓扑 2](#_Toc134285736)

[在拓扑中SDC ping MIT建立连接 3](#_Toc134285737)

[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 3](#_Toc134285738)

[基础部分实验 4](#_Toc134285739)

[输出每次影响EC的数量 4](#_Toc134285740)

[输出环路路径 4](#_Toc134285741)

[进一步打印出环路对应的EC的相关信息 4](#_Toc134285742)

[分析原始代码与补丁代码的区别，思考为何需要添加补丁 5](#_Toc134285743)

[拓展部分实验 7](#_Toc134285744)

[若修改waypoint\_path.py代码中被添加规则的优先级字段，VeriFlow的检测结果会出错，试描述错误是什么，并解释出错的原因 7](#_Toc134285745)

[在其支持的域中，选择几个域来进行验证。 8](#_Toc134285746)

## 实验要求

### 1.基础实验部分

输出每次影响EC的数量

打印出环路路径的信息

进一步打印出环路对应的EC的相关信息

分析原始代码与补丁代码的区别，思考为何需要添加补丁

### 2.拓展实验部分

若修改waypoint\_path.py代码中被添加规则的优先级字段，VeriFlow的检测结果会出错，试描述错误是什么，并解释出错的原因

在VeriFlow支持的14个域中，挑选多个域（不少于5个）进行验证，输出并分析结果

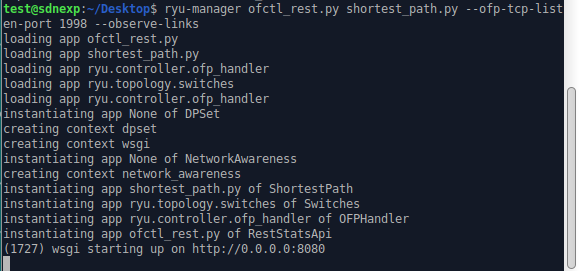
## 准备部分

### 安装veriFlow并编译

git clone https://github.com/samueljero/BEADS.git  
cd BEADS  
git am 0001-for-xjtu-sdn-exp-2020.patch

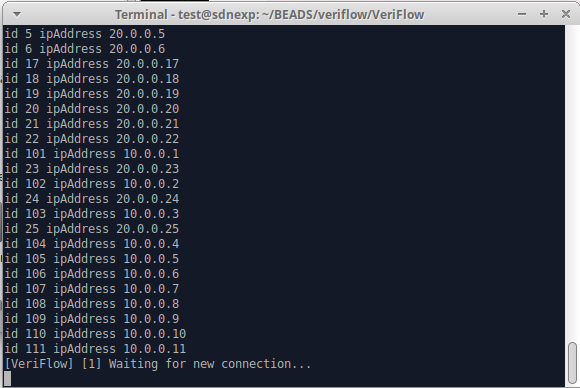
### 在自定义端口开启远程控制器，运行最短路程序

ryu-manager ofctl\_rest.py shortest\_path.py --ofp-tcp-listen-port 1024 --observe-links



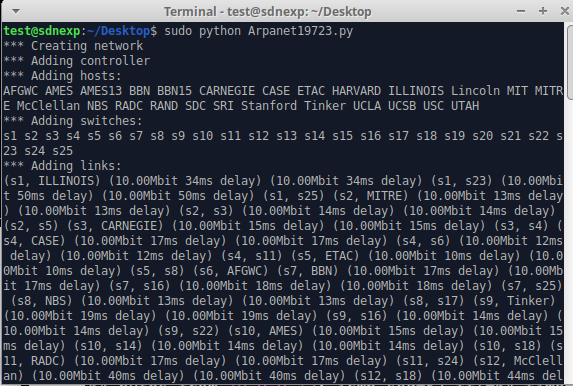
### 运行VeriFlow的proxy模式

./VeriFlow 6633 127.0.0.1 1024 Arpanet19723.txt log\_file.txt



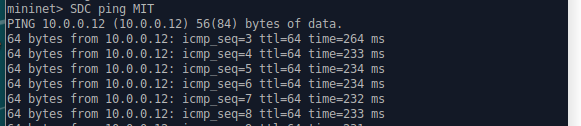
### 启动拓扑

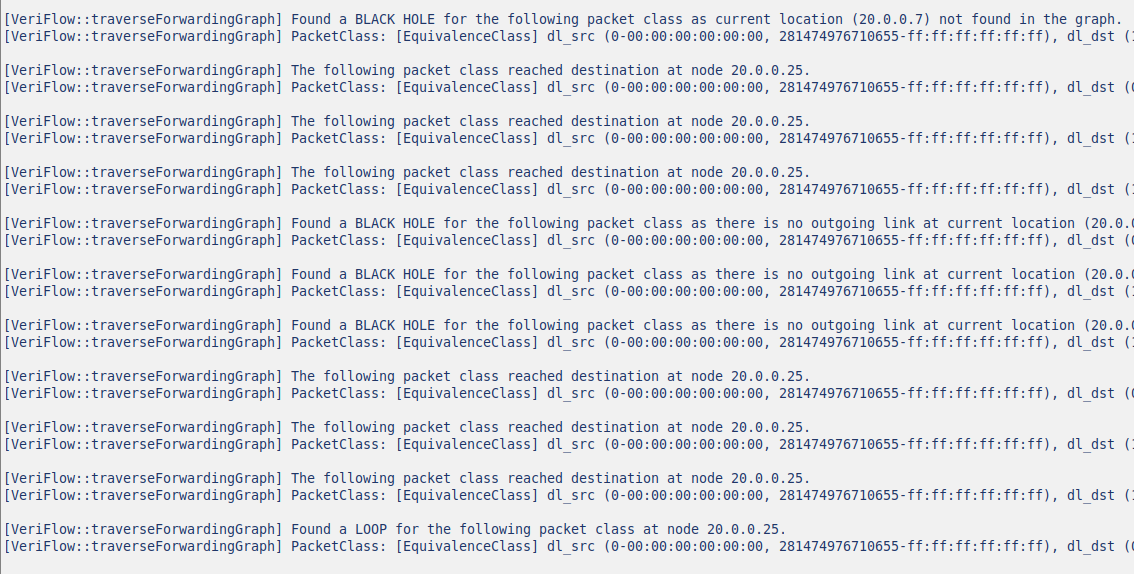
sudo python Arpanet19723.py



### 在拓扑中SDC ping MIT建立连接

SDC ping MIT



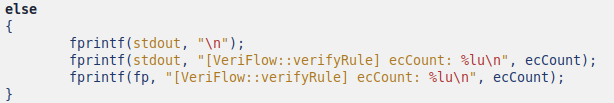
下发从UTAH途经TINKER到达ILLINOIS的路径，在log文件中观察VeriFlow检测到的环路信息  
sudo python waypoint\_path.py



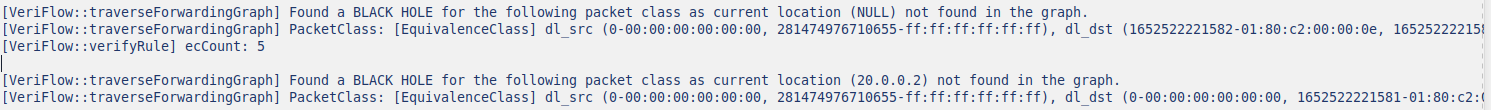
## 基础部分实验

### 输出每次影响EC的数量

将veriflow中verifyRule()的原代码更改为如下所示：



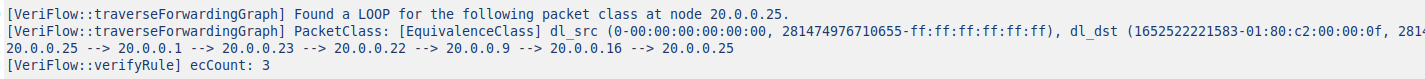
之后重新编译可观察到输出日志中出现数量：



### 输出环路路径

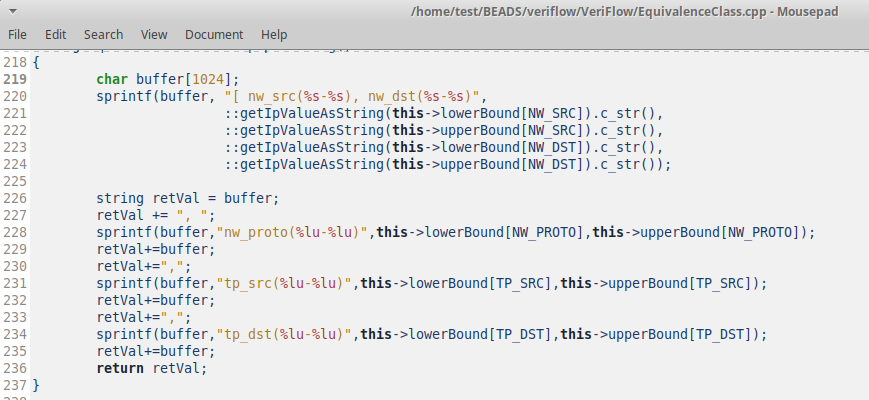
将veriflow.cpp与veriflow.h文件中的traverseForwardingGraph函数中的visited改为vector类型，之后在每次输出时输出其中的路径即可。

下图为重新编译后的结果。

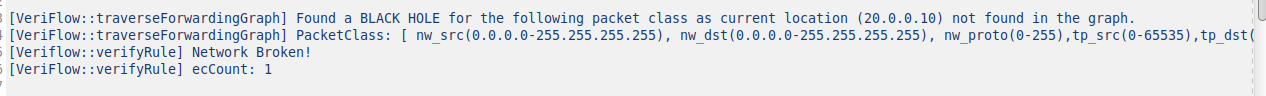


### 进一步打印出环路对应的EC的相关信息

通过查看代码可以得出在traverseForwardingGraph函数中是通过调用equivalenceClass文件中的toString函数来进行输出的，故需将其更改为如下形式：



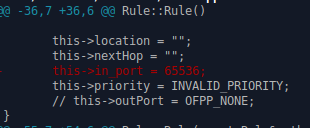
这样可以使其仅输出与tcp或ip有关的五元组。



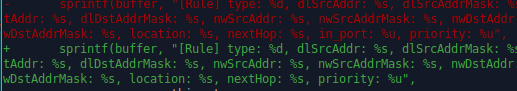
### 分析原始代码与补丁代码的区别，思考为何需要添加补丁

经搜索可得我们可以使用git中的diff参数来实现对比原始版本与修改后版本。

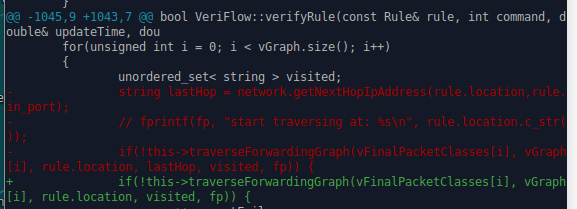
可以发现在Rule类中添加了in\_port参数；



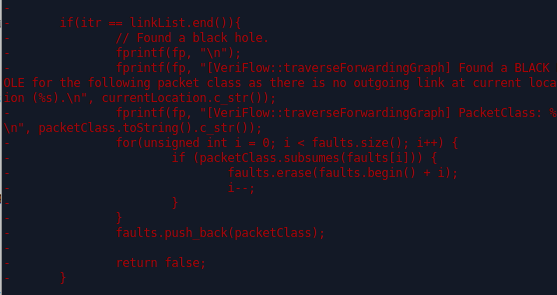
并对其进行了处理，如下图中对其的输出等操作：



在veriflow的verifyRole中对lastHop进行存储：



经查可以提升查找BlackHole的能力。

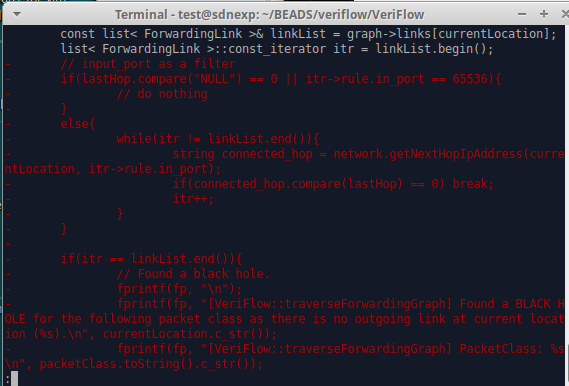


原理如下：

补丁中增加了一种判断黑洞的情况：当前的交换机或者主机在网络的拓扑结构中，也存在与它相连的链路，但由于网络结构变化，使得从当前的交换机或者主机所在位置和相应端口(in\_port)，找不到上一跳的交换机或者主机。

完善了对环路的判断

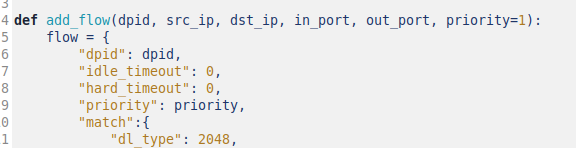
选择下一跳避免了与上一跳相同



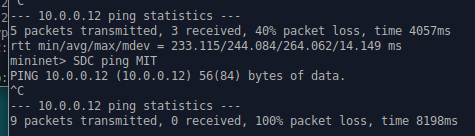
## 拓展部分实验

### 若修改waypoint\_path.py代码中被添加规则的优先级字段，VeriFlow的检测结果会出错，试描述错误是什么，并解释出错的原因

修改waypoint\_path.py代码中被添加规则的优先级字段改为1



并运行，可以观察到日志文件中无环路，且SDC ping MIT不通



原因是在更改优先级后，当匹配域相同时，在waypoint\_path代码中下发的流表项优先级与原流表项一致并更新，故会将原流表项覆盖掉。

判断环路选择下一跳时，veriflow利用priority字段进行了排序（即graph->links[currentLocation].sort(compareForwardingLink)语句），当出现完全匹配、priority相同的规则，并需要对原来的规则进行覆盖的时候， VeriFlow 并没有将原来的规则删除并加上新的规则，而是保留了原来的规则并抛弃了新加的规则。故其无法检测环路。

### 在其支持的域中，选择几个域来进行验证。

出于简单与实用考虑，本次实验选择原与目的IP地址、MAC地址与入端口来进行验证。（局域网、两种协议类型与目的与原端口、MPLS较之较少使用，MATADATA域不使用）。

像waypoint\_path中添加原与目的的MAC地址（即数据链路层地址），并进行验证。



可以看见在更改后，EC数目为3项，与基础部分实验数目相同，并出现环路，与预期一致。

