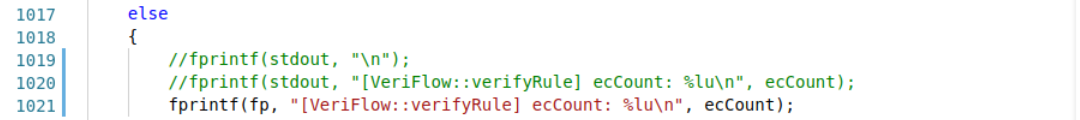
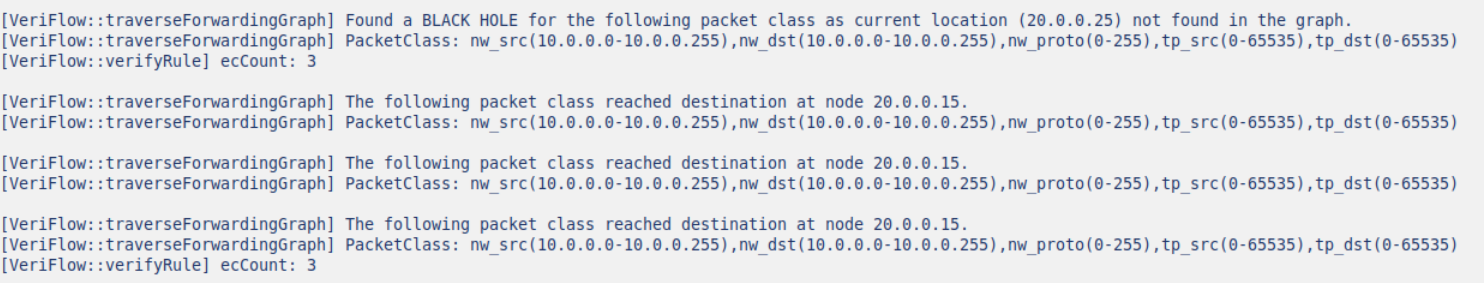
**SDN实验四**

1. 输出影响的等价类的数量：

在VeriFlow.cpp中的verifyRule函数中作如下修改，重新make编译后可以在log\_file中看到输出的ecCount



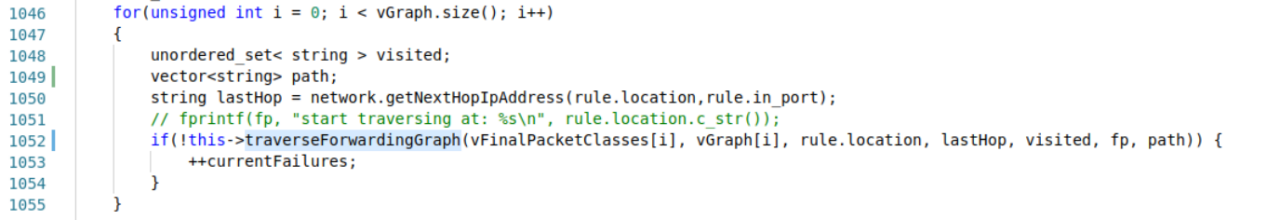


1. 打印环路路径信息：

首先在VeriFlow.h中修改traverseForwadingGraph的定义，加上一个vector数组保存路径信息

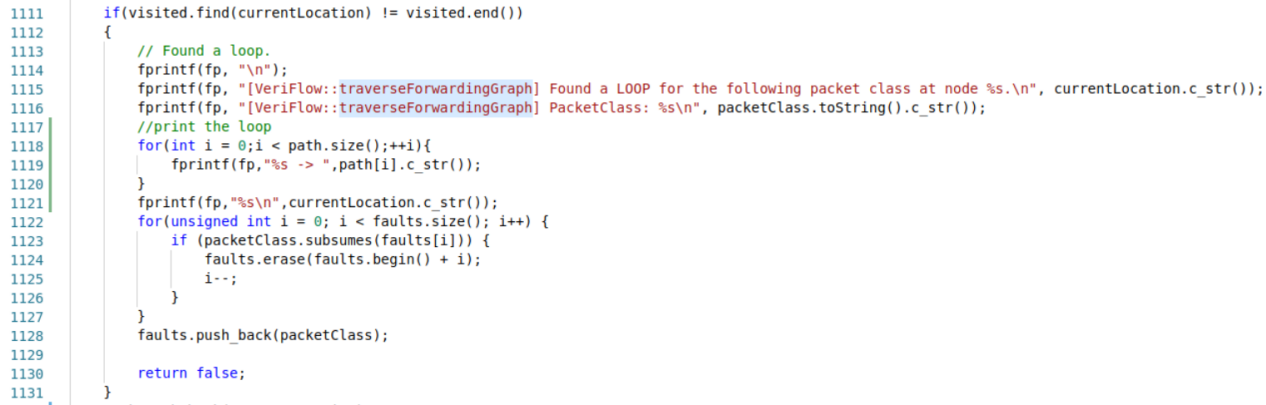


在VeriFlow.cpp中的verifyRule函数中添加一个字符串数组

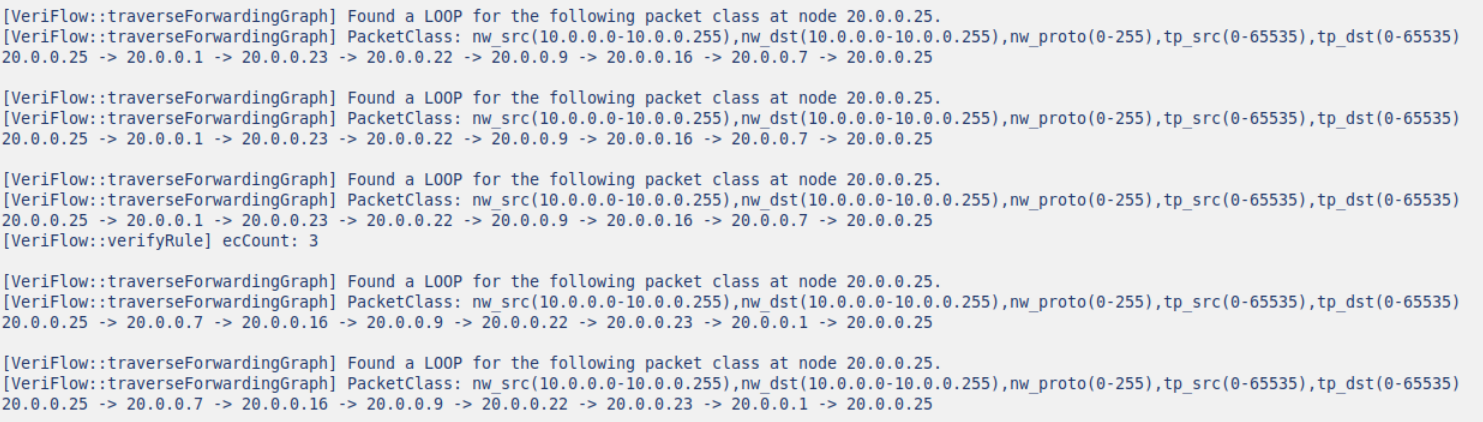


在VeriFlow.cpp中的traverseForwadingGraph函数中加入记录路径和输出环路路径的代码



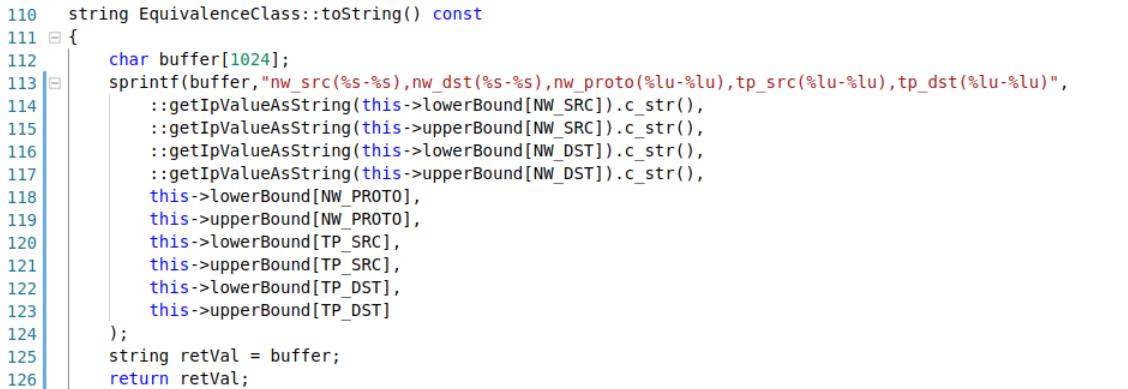


完成后输出的环路路径如下所示

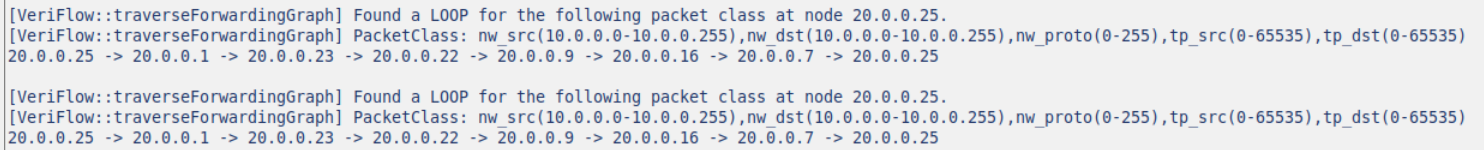


1. 进一步打印出环路对应的EC的相关信息

EC的信息输出是通过to\_String()实现的，因此在EquivalenceClass.cpp中修改to\_String函数即可，从中提取出所需的tcp/ip五元组



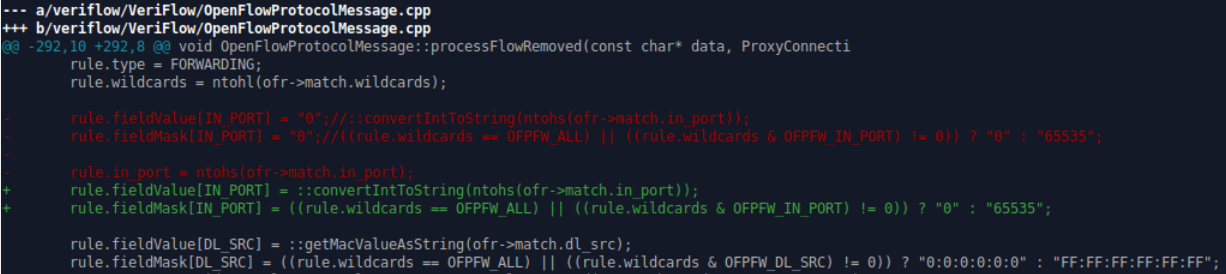
重新编译后的输出

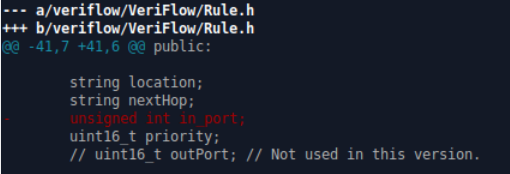


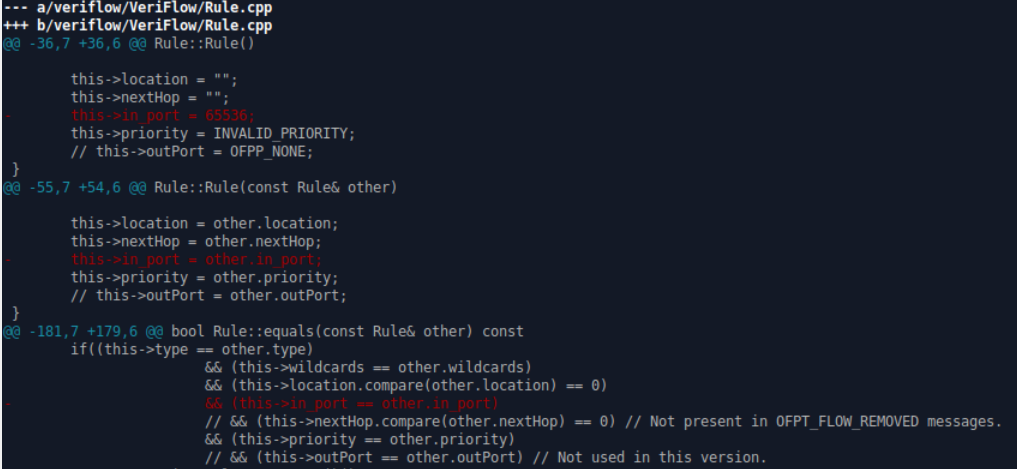
1. 分析原始代码与补丁代码的区别，思考为何需要添加补丁

原始代码和补丁代码的区别：

1. 补丁代码中将rule中的fieldValue[IN\_PORT]置为0，同时在Class Rule中增加了一个in\_port属性，增加了对in\_port的存储和处理。

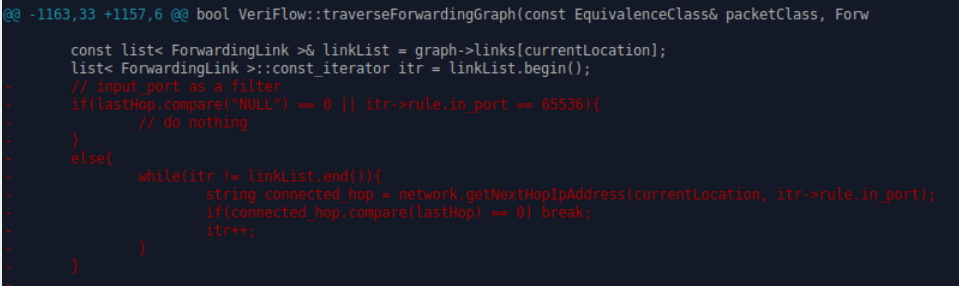






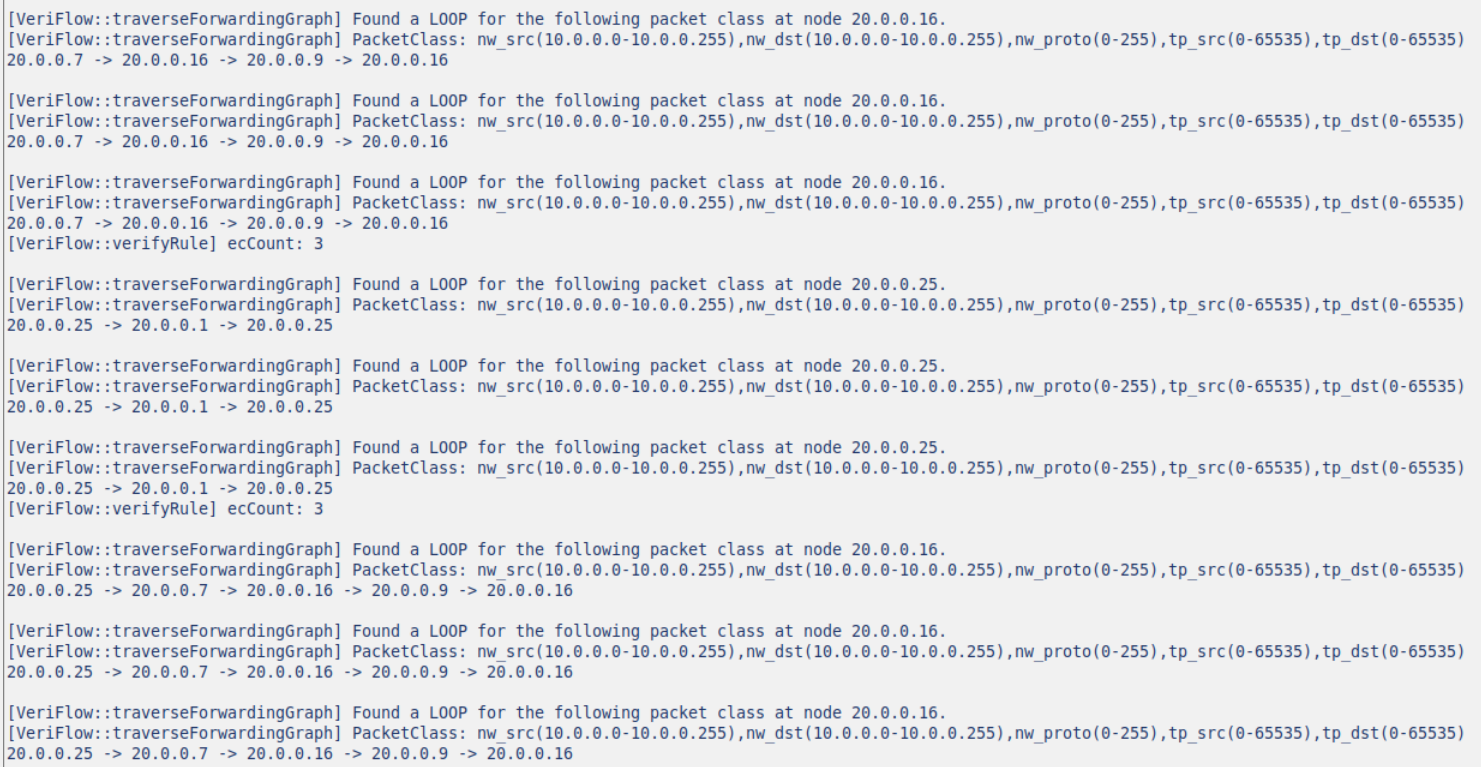
而原始代码中在计算EC时考虑了in\_port，这是不对的，因为在数据包转发过程中，不能保证每次进入的in\_port是相同的，考虑了in\_port会导致EC数量过多。

1. 补丁代码完善了对环路的检测，在traverseForwardingGraph函数中增加了lastHop参数

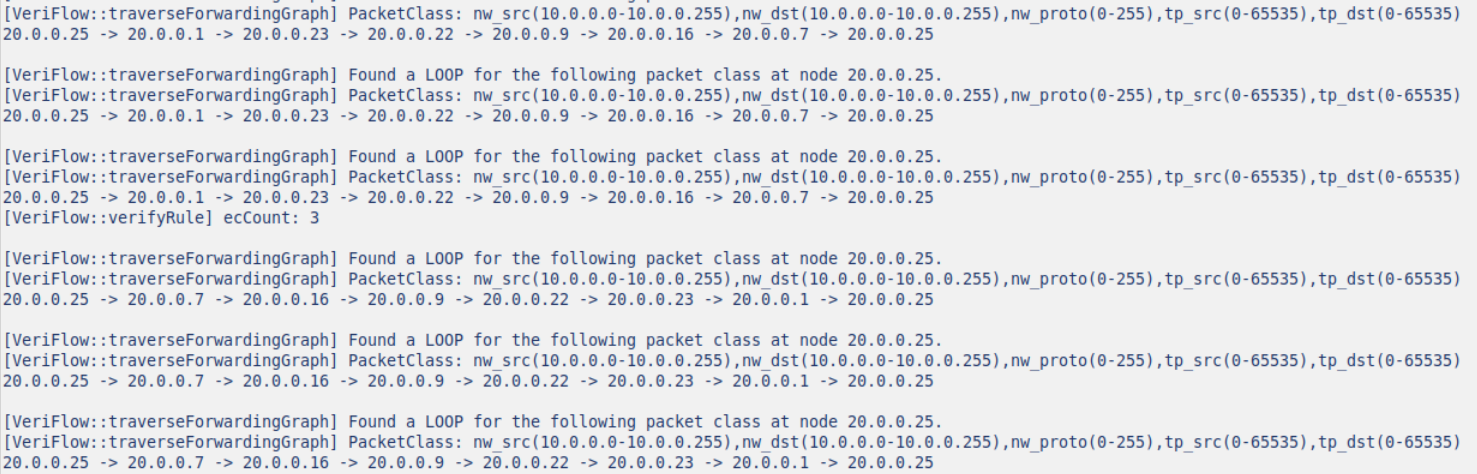


原始代码和补丁代码运行结果对比：

原始代码：



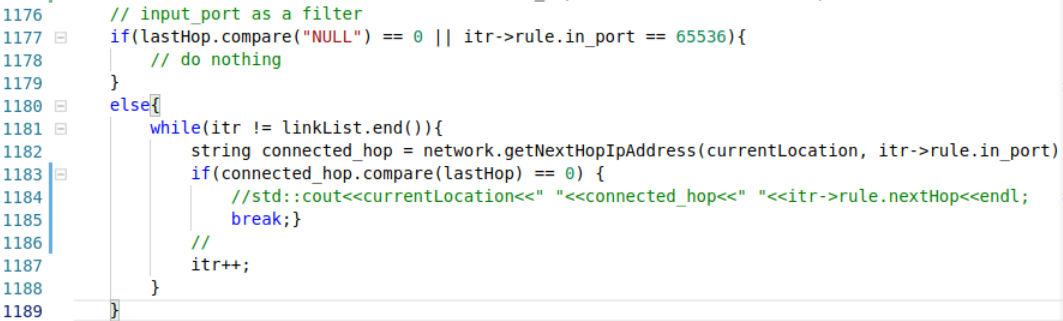
补丁代码：



可以发现，在原始代码中，检测出的环路仅有两个节点，如20.0.0.25->20.0.0.1->20.0.0.25，或20.0.0.16->20.0.0.9->20.0.0.16，这是因为在原始代码中，只有下面这两行代码，itr永远是linkList中的第一项



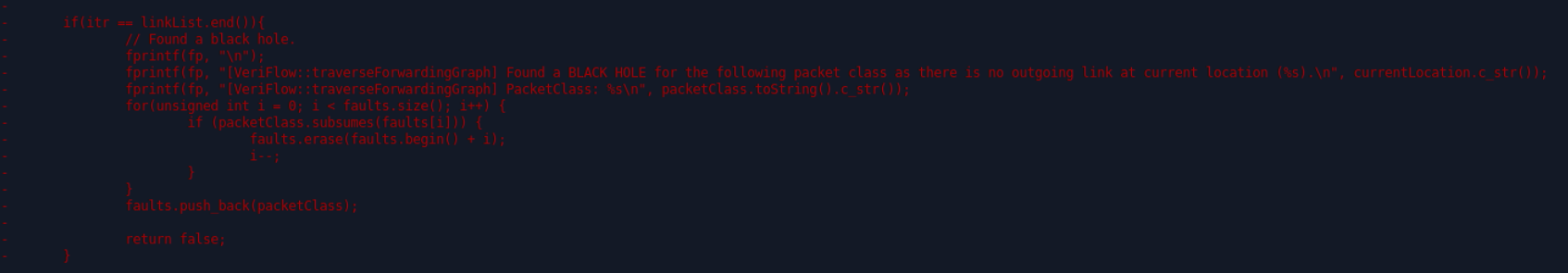
之后将itr->rule.nextHop作为currentLocation进行迭代，而这个itr->rule.nextHop可能与lastHop相同，从而被判定为环路。

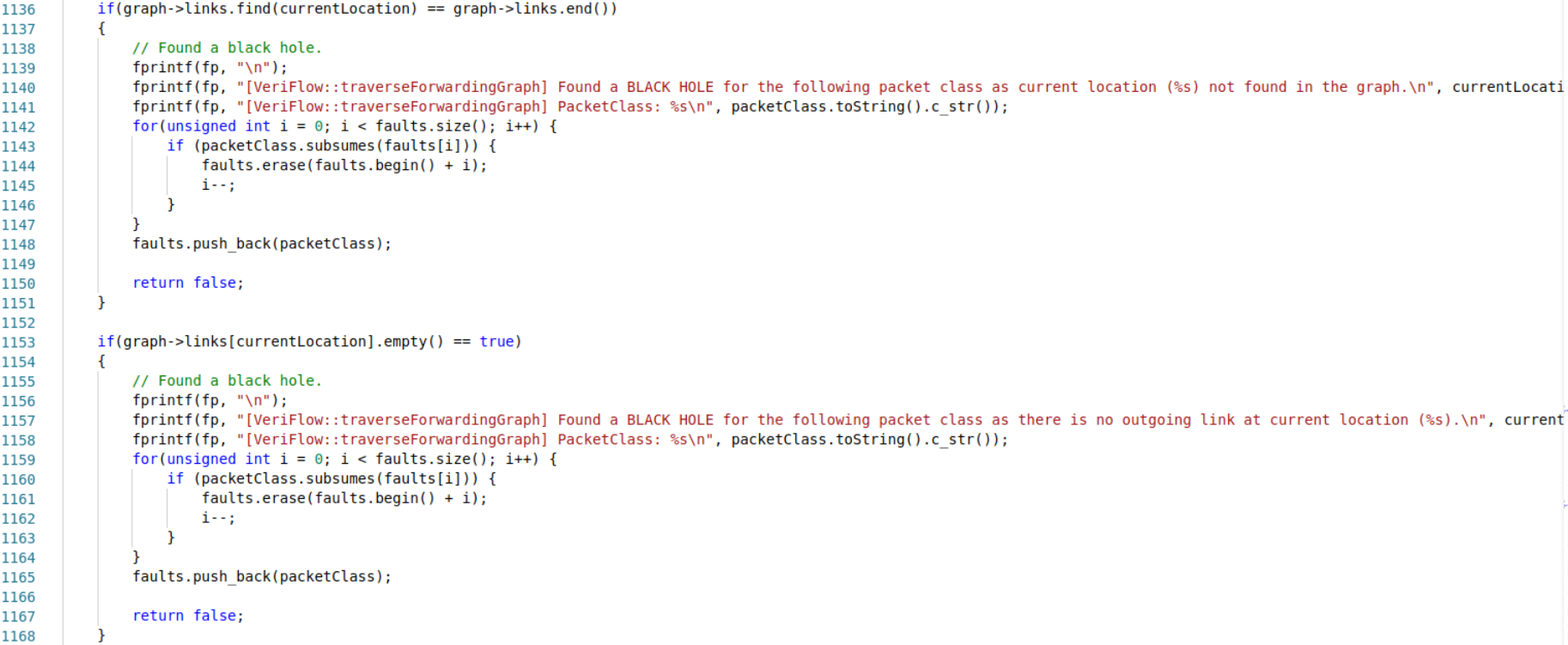


补丁代码中加入了这一段代码，在while循环中，根据currentLocation和in\_port计算出与当前节点相连的节点的ip地址，如果与lastHop的ip地址相同，即说明当前的itr是数据包正确的转发路径，此时再按照itr->rule.nextHop进行迭代即可得到正确的环路。

1. 补丁代码完善了对黑洞的检测，通过记录lastHop来检测黑洞

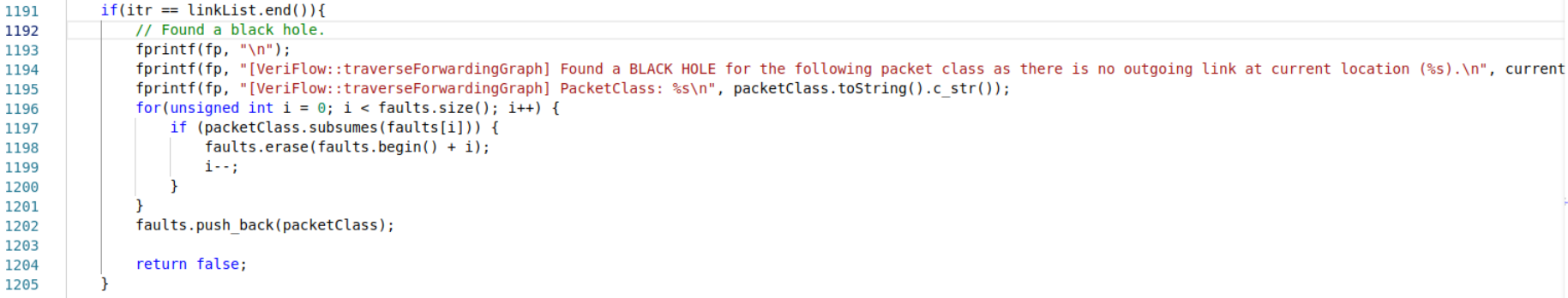
补充的代码：





原始代码中只有两种判断黑洞的方法：

1. 当前主机或交换机不在网络中；
2. 当前主机或交换机在网络中，但没有链路和其它交换机或主机相连。



补丁代码补充了一种情况，当前的主机或交换机在网络中，并且有链路与其他的主机或交换机相连，但由于某些变化导致当前主机或交换机找不到上一跳的主机或交换机，这种情况也会被判为黑洞。