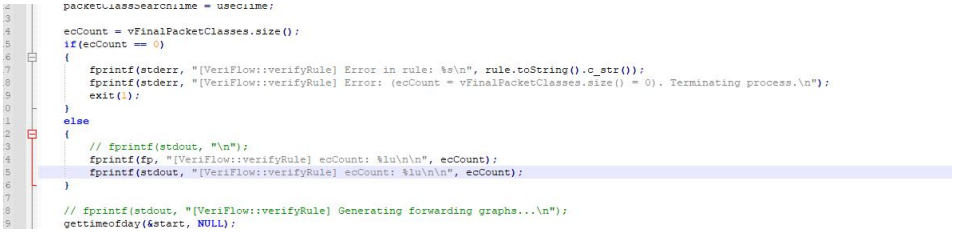
第四次实验报告

1. EC 数量

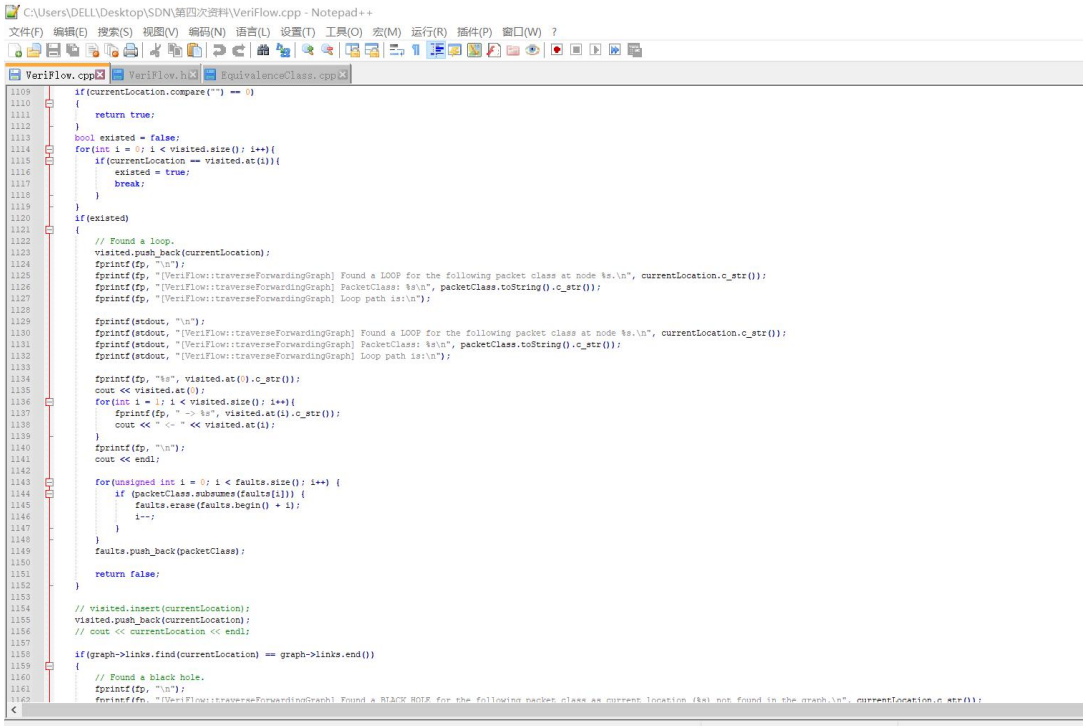
根据文件的提示，可以看到在处理验证规则的函数 verifyRule 中，每个规则的 ecCount 的数目是在函数的开始计算的，在计算结束之后输出到文件流中即可。



```
packetClassSearchTime = useciime;

ecCount = vFinalPacketClasses.size();
if(ecCount == 0)
{
    fprintf(stderr, "[VeriFlow::verifyRule] Error in rule: %s\n", rule.toString().c_str());
    fprintf(stderr, "[VeriFlow::verifyRule] Error: (ecCount = vFinalPacketClasses.size() = 0). Terminating process.\n");
    exit(1);
}
else
{
    // fprintf(stdout, "\n");
    fprintf(fp, "[VeriFlow::verifyRule] ecCount: %lu\n\n", ecCount);
    fprintf(stdout, "[VeriFlow::verifyRule] ecCount: %lu\n\n", ecCount);
}

// fprintf(stdout, "[VeriFlow::verifyRule] Generating forwarding graphs...\n");
gettimeofday(&start, NULL);
```

2. 环路输出

打印出环路信息：首先锁定 traverseForwardingGraph 函数，该函数用来遍历一个固定的 EC 集合并计算转发图，如果找到环路之后就打印相关信息。这里看了下 unordered_set 的作用，主要是用来判断第一个出现重复元素的地方，说明在这之前的函数的递归调用时没有重复元素的，所以可以用 vector 等动态数组来记录插入时的顺序



这里的环路的最终的结点没有正常打印，这是因为在 trans 函数中插入节点的位置存在问题，在函数调用的最开始插入数据即可显示出完整的环路信息。

```
bool verifyRule(const Rule& rule, int command, double& updateTime, double& packetClassSearchTime, double& graphBuildTime, double& queryTime, unsigned long& ecCount, FILE* fp);
bool traverseForwardingGraph(const EquivalenceClass& packetClass, ForwardingGraph* graph, const string& currentLocation, const string& lastHop, vector < string > visited, FILE* fp);

int getTotalRuleCount() const;

void setDatapathId(unsigned short socketPort, uint64_t datapathId);
```

3. 五元组打印

五元组信息打印：这里主要是通过修改 EC 等价类的 to_string 函数进行打印，涉及到对应的 EquivalenceClass.cpp 中函数，主要是过滤出来需要的五元组即可。

```
109
110 string EquivalenceClass::toString() const
111 {
112     char buffer[1024];
113     sprintf(buffer, "nw_src (%s - %s), nw_dst (%s - %s)",
114             ::getIpValueAsString(this->lowerBound[NW_SRC]).c_str(),
115             ::getIpValueAsString(this->upperBound[NW_SRC]).c_str(),
116             ::getIpValueAsString(this->lowerBound[NW_DST]).c_str(),
117             ::getIpValueAsString(this->upperBound[NW_DST]).c_str());
118     string retVal = buffer;
119     retVal += ", ";
120
121     sprintf(buffer, "nw_proto (%lu - %lu), tp_src (%lu - %lu), tp_dst (%lu - %lu)",
122             this->lowerBound[10], this->upperBound[10],
123             this->lowerBound[12], this->upperBound[12],
124             this->lowerBound[13], this->upperBound[13]);
125
126     retVal += buffer;
127
128     return retVal;
129     // char buffer[1024];
130     // sprintf(buffer, "[EquivalenceClass] dl_src (%lu-%s, %lu-%s), dl_dst (%lu-%s, %lu-%s)",
131     //         this->lowerBound[DL_SRC], ::getMacValueAsString(this->lowerBound[DL_SRC]).c_str(),
132     //         this->upperBound[DL_SRC], ::getMacValueAsString(this->upperBound[DL_SRC]).c_str(),
```

实验四:

这里可以看到，主要的变化是对规则的输入端口的设置以及对应在比较时增加对端口的比较，对每个规则都增加相应的输入端口的过滤来进一步处理。通过对比在控制台的输出，可以看到增加补丁之后的faults size 是变小的，可以认为是新的补丁可以减少对于错误的判断，只修复关键的节点，减少消耗。

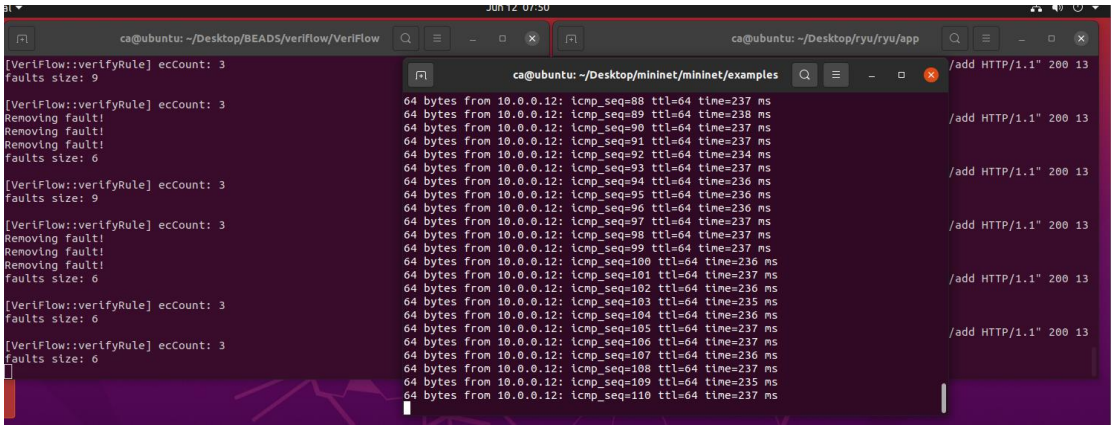
```
@@ -292,10 +292,8 @@ void OpenFlowProtocolMessage::processFlowRemoved(const char* data, ProxyConnecti
         rule.type = FORWARDING;
         rule.wildcards = ntohl(ofr->match.wildcards);

-        rule.fieldValue[IN_PORT] = "0";//::convertIntToString(ntohs(ofr->match.in_port));
-        rule.fieldMask[IN_PORT] = "0";//((rule.wildcards == OFPFW_ALL) || ((rule.wildcards & OFPFW_IN_PORT) != 0)) ? "0" : "65535";
-
-        rule.in_port = ntohs(ofr->match.in_port);
+        rule.fieldValue[IN_PORT] = ::convertIntToString(ntohs(ofr->match.in_port));
+        rule.fieldMask[IN_PORT] = ((rule.wildcards == OFPFW_ALL) || ((rule.wildcards & OFPFW_IN_PORT) != 0)) ? "0" : "65535";

         rule.fieldValue[DL_SRC] = ::getMacValueAsString(ofr->match.dl_src);
         rule.fieldMask[DL_SRC] = ((rule.wildcards == OFPFW_ALL) || ((rule.wildcards & OFPFW_DL_SRC) != 0)) ? "0:0:0:0:0:0" : "FF:FF:FF:FF:FF:FF";
@@ -348,13 +346,12 @@ void OpenFlowProtocolMessage::processFlowRemoved(const char* data, ProxyConnecti
```

```
@@ -55,7 +54,6 @@ Rule::Rule(const Rule& other)

     this->location = other.location;
     this->nextHop = other.nextHop;
-    this->in_port = other.in_port;
     this->priority = other.priority;
     // this->outPort = other.outPort;
 }
@@ -181,7 +179,6 @@ bool Rule::equals(const Rule& other) const
     if((this->type == other.type)
                 && (this->wildcards == other.wildcards)
                 && (this->location.compare(other.location) == 0)
-                && (this->in_port == other.in_port)
                 // && (this->nextHop.compare(other.nextHop) == 0) // Not present in OFPT_FLOW_REMOVED messages.
                 && (this->priority == other.priority)
                 // && (this->outPort == other.outPort) // Not used in this version.
@@ -225,7 +222,6 @@ int Rule::operator()() const
```

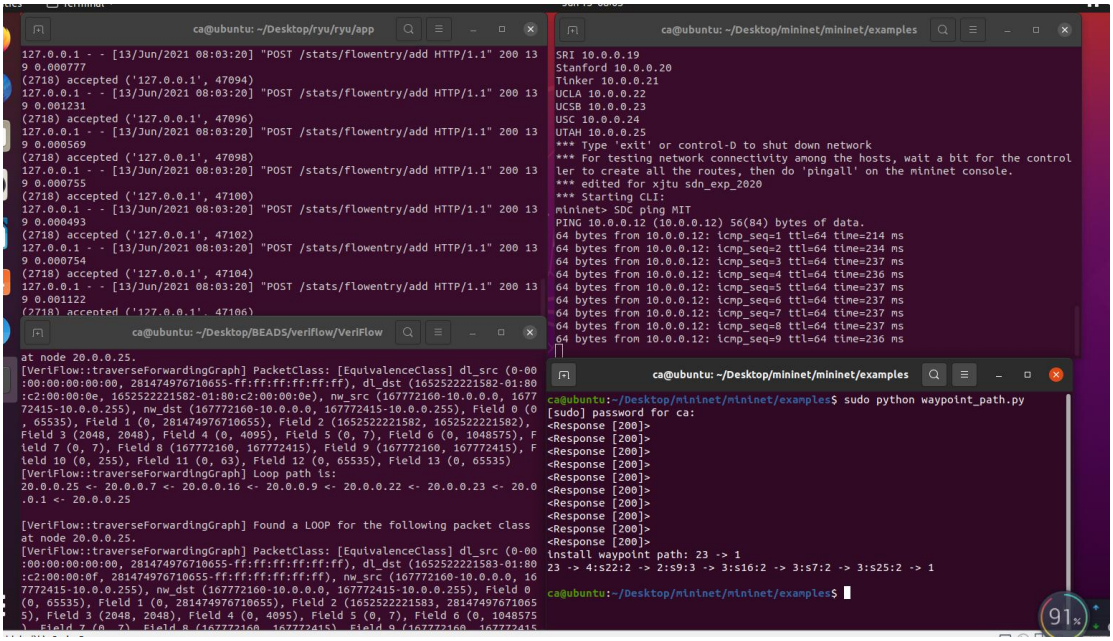
```
    const list< ForwardingLink >& linkList = graph->links[currentLocation];
    list< ForwardingLink >::const_iterator itr = linkList.begin();
    // input_port as a filter
    if(lastHop.compare("NULL") == 0 || itr->rule.in_port == 65536){
            // do nothing
    }
    else{
            while(itr != linkList.end()){
                    string connected_hop = network.getNextHopIpAddress(currentLocation, itr->rule.in_port);
                    if(connected_hop.compare(lastHop) == 0) break;
                    itr++;
            }
    }

    if(itr == linkList.end()){
            // Found a black hole.
            fprintf(fp, "\n");
            fprintf(fp, "[VeriFlow::traverseForwardingGraph] Found a BLACK HOLE for the following packet class as there is no outgoing link at current location (%s).\
tLocation.c_str());
            fprintf(fp, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str());
            for(unsigned int i = 0; i < faults.size(); i++) {
                    if (packetClass.subsumes(faults[i])) {
                            faults.erase(faults.begin() + i);
                            i--;
                    }
            }
            faults.push_back(packetClass);

            return false;
    }

    if(itr->isGateway == true)
    {
@@ -1220,7 +1187,7 @@ bool VeriFlow::traverseForwardingGraph(const EquivalenceClass& packetClass, Forw
                    fprintf(fp, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str()); */
            }

            return this->traverseForwardingGraph(packetClass, graph, itr->rule.nextHop, currentLocation, visited, fp);
            return this->traverseForwardingGraph(packetClass, graph, itr->rule.nextHop, visited, fp);
    }
```

结果展示:

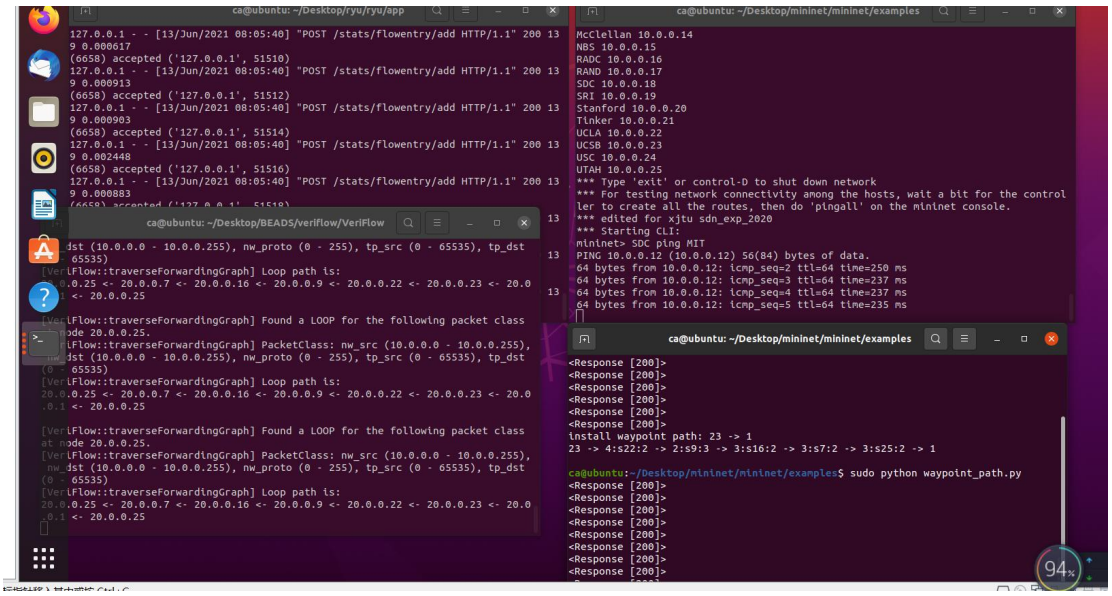
实验一:



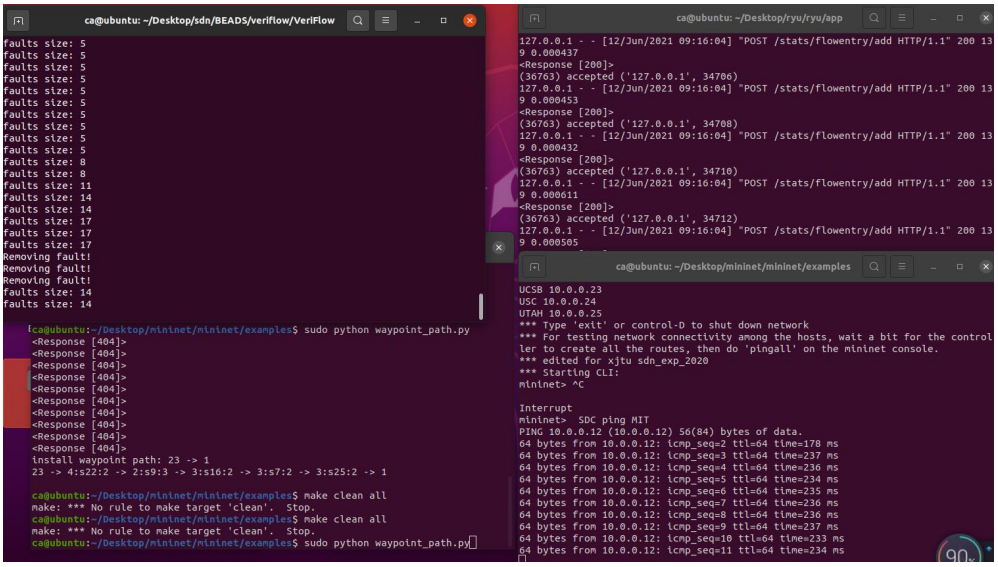
实验二:


ca@ubuntu: ~/Desktop/ryu/ryu/app
ca@ubuntu: ~/Desktop/mininet/mininet/examples
ca@ubuntu: ~/Desktop/BEADS/veriflow/VeriFlow
ca@ubuntu: ~/Desktop/mininet/mininet/examples
ca@ubuntu:~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py
[sudo] password for ca:
install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1

实验三:


ca@ubuntu: ~/Desktop/ryu/ryu/app
ca@ubuntu: ~/Desktop/mininet/mininet/examples
ca@ubuntu: ~/Desktop/BEADS/veriflow/VeriFlow
ca@ubuntu: ~/Desktop/mininet/mininet/examples
ca@ubuntu:~/Desktop/mininet/mininet/examples$ 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

实验四:


ca@ubuntu: ~/Desktop/sdn/BEADS/veriflow/VeriFlow
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 8
faults size: 8
faults size: 11
faults size: 14
faults size: 14
faults size: 17
faults size: 17
faults size: 17
Removing fault!
Removing fault!
Removing fault!
faults size: 14
faults size: 14
ca@ubuntu:~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1
ca@ubuntu:~/Desktop/mininet/mininet/examples$ make clean all
make: *** No rule to make target 'clean'. Stop.
ca@ubuntu:~/Desktop/mininet/mininet/examples$ make clean all
make: *** No rule to make target 'clean'. Stop.
ca@ubuntu:~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py
ca@ubuntu: ~/Desktop/ryu/ryu/app
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000437
<Response [200]>
(36763) accepted ('127.0.0.1', 34706)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000453
<Response [200]>
(36763) accepted ('127.0.0.1', 34708)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000432
<Response [200]>
(36763) accepted ('127.0.0.1', 34710)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000611
<Response [200]>
(36763) accepted ('127.0.0.1', 34712)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000505
ca@ubuntu: ~/Desktop/mininet/mininet/examples
UCSB 10.0.0.23
USC 10.0.0.24
UTAH 10.0.0.25
*** Type 'exit' or control-D to shut down network
*** For testing network connectivity among the hosts, wait a bit for the control
ler to create all the routes, then do 'pingall' on the mininet console.
*** edited for xjtu sdn_exp_2020
*** Starting CLI:
mininet> ^C
Interrupt
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=178 ms
64 bytes from 10.0.0.12: icmp_seq=3 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=4 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=5 ttl=64 time=234 ms
64 bytes from 10.0.0.12: icmp_seq=6 ttl=64 time=235 ms
64 bytes from 10.0.0.12: icmp_seq=7 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=8 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=9 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=10 ttl=64 time=233 ms
64 bytes from 10.0.0.12: icmp_seq=11 ttl=64 time=234 ms